 [Translated from Vietnamese to English - www.onlinedoctranslator.com](https://www.onlinedoctranslator.com/en/?utm_source=onlinedoctranslator&utm_medium=docx&utm_campaign=attribution" \o "Doc Translator - www.onlinedoctranslator.com)



**FACULTY OF COMPUTER SCIENCE & ENGINEERING**

**PROJECT REPORT**

**OBJECT ORIENTED PROGRAMMING (OOP) COURSE**

**TOPIC**

**MARKETPLACE TRANSACTIONAL**

**STUDENT**

DOAN TRONG DINH – 77482403176 – K17

NGUYEN DUY QUY – 77482403203 – K17

**LECTURER:**Dr. LE NGOC HIEU

SEMESTER 2, 2024-2025

**SCORE RULE**

Full name of student 1: ……..DOAN TRONG DINH…….. Class: 24CVM

Full name of student 2: ……..NGUYEN DUY QUY……..Class: 24CVM

Project/Thesis Name:

….……………….……….MarketplaceTransactional………………………..

Evaluate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TT | Criteria | Scoring | Dot | Note |
| 1 | Presentation format:  - Present according to instructions (font, page numbers, table of contents, tables, reference list, etc.).  - No spelling mistakes, typos, or reference errors.  - Beautiful presentation, clear writing style, not obscure. |  |  |  |
| 2 | Content | | | |
| Chapter 1: Theoretical basis and research status |  |  |  |
| Chapter 2: Solution and application design analysis |  |  |  |
| Chapter 3: How the system works |  |  |  |
| Chapter 4: Product testing and evaluation |  |  |  |
| Chapter 5 |  |  |  |
| 3 | TOTAL SCORE | |  |  |

Letter grade: ………………………………………… (Rounded to 1 decimal place)

# **INDEX**

[Scorecard i](#_Toc114824147)

Table of Contents................................................................................................ii

[List of terms, symbols, abbreviations ii](#_Toc114824148)

[List of tables iii](#_Toc114824149)

[List of figures, graphs iv](#_Toc114824150)

[INTRODUCTION 1](#_Toc114824151)

**[Chapter 1 . Theoretical basis and research status 2](#_Toc114824152)**

[1.1](#_Toc114824153) **[Scientific basis](#_Toc114824153)** [2](#_Toc114824153)

[1.1.1 Object Oriented Programming (OOP) and its role in the system 2](#_Toc114824154)

[1.1.2 JavaFX – A Platform for Building Modern User Interfaces 2](#_Toc114824155)

[1.1.3 MySQL Database – Efficiently Manage Transaction Information 2](#_Toc114824156)

[1.1.4 Algorithms and data structures in business processing 2](#_Toc114824156)

[1.1.5 JFreeChart and the role of data visualization 2](#_Toc114824156)

[1.1.6 Security and user authentication 2](#_Toc114824156)

[1.2](#_Toc114824157) **[Current status and research orientation](#_Toc114824157)** [2](#_Toc114824157)

[1.2.1 Current status of e-commerce transaction system development 2](#_Toc114824158)

[1.2.2 Expansion orientation 2](#_Toc114824159)

[1.3 Third item 2](#_Toc114824160)

**[Chapter 2. Application design analysis and solutions 3](#_Toc114824161)**

[2.1 System architecture overview 2](#_Toc114824153)

[2.2 Detailed design of components 2](#_Toc114824153)

[2.3 Main business process 2](#_Toc114824153)

[2.4 Security Risk Analysis & Solutions – Expand Details 2](#_Toc114824153)

[2.5 Detailed test plan 2](#_Toc114824153)

[2.6 Implementation and Operation Plan 2](#_Toc114824153)

[2.7 User Interface Design Analysis 2](#_Toc114824153)

**[Chapter 3 . How the system works 4](#_Toc114824162)**

[3.1 Overall architecture of the system 2](#_Toc114824153)

[3.2 User authentication and authorization mechanism 2](#_Toc114824153)

[3.3 Display products and manage shopping cart 2](#_Toc114824153)

[3.4 Payment and order processing mechanism 2](#_Toc114824153)

[3.5 Enhanced features and experience 2](#_Toc114824153)

[3.6 System statistics and administration dashboard 2](#_Toc114824153)

[3.7 Database and transaction management 2](#_Toc114824153)

[3.8 System scalability 2](#_Toc114824153)

**[Chapter 4 . Product testing and evaluation 4](#_Toc114824162)**

[4.1 Experimental objectives 2](#_Toc114824153)

[4.2 Experimental environment 2](#_Toc114824153)

[4.3 Experimental procedure 2](#_Toc114824153)

[4.4 Performance Evaluation 2](#_Toc114824153)

[4.5 User Feedback 2](#_Toc114824153)

[4.6 Summary of experimental results 2](#_Toc114824153)

[CONCLUDE 5](#_Toc114824163)

[REFERENCES 6](#_Toc114824164)

# **LIST OF SYMBOLS AND ABBREVIATIONS**

| **STT** | **Abbreviations/Symbols** | **English origin** | **Vietnamese translation/caption** |
| --- | --- | --- | --- |
| 1 | Database | Database(s) | Database |
| 2 | I | Items | Set of items |
| 3 | OOP | Object-Oriented Programming | Object Oriented Programming |
| 4 | API | Application Programming Interface | Application Programming Interface |
| 5 | CRUD | Create, Read, Update, Delete | Create, Read, Update, Delete |
| 6 | CSS | Cascading Style Sheets | Cascading stylesheet |
| 7 | KNIFE | Data Access Object | Data access object |
| 8 | DB | Database | Database |
| 9 | ERD | Entity-Relationship Diagram | Entity-Relationship Diagram |
| 10 | FXML | FX Markup Language | Markup Language for JavaFX |
| 11 | GUI | Graphical User Interface | Graphical User Interface |
| 12 | IDE | Integrated Development Environment | Integrated Development Environment |
| 13 | JDBC | Java Database Connectivity | Database Connection in Java |
| 14 | JDK | Java Development Kit | Java Development Kit |

# **LIST OF FIGURES AND GRAPHICS**

[Figure 1.1. Overview model 2](#_Toc278635328)

[Figure 1.2.main home screen 2](#_Toc278635329)

[Figure 1.3 login screen 2](#_Toc278635330)

[Figure 1.4.registration screen 2](#_Toc278635331)

[Figure 1.5.Product List 2](#_Toc278635332)

[Figure 1.6.shopping cart screen 2](#_Toc278635333)

[Figure 1.7payment screen 2](#_Toc278635334)

[Figure 1.8for admin 2](#_Toc278635335)

[Figure 1.9Order history 2](#_Toc278635335)

[Figure 2.0list of favorite products 2](#_Toc278635335)

[Figure 2.1 support chat 2](#_Toc278635335)

**SUMMARY OF THESIS**

**1. Background and reasons for choosing the topic**

The rapid development of digital technology has strongly promoted the digitalization process in the field of commerce. E-commerce platforms are becoming the main sales channel for many individuals and businesses. Names such as Shopee, Lazada, Tiki, Amazon,... not only act as a place to display products but also act as a transaction intermediary (marketplace) between buyers and sellers.

In this model, transactions are the central component – ​​where ordering, payment, shipping, etc. take place. Therefore, collecting, managing, analyzing and visualizing transaction data becomes extremely important. Not only does it help businesses understand customer behavior, but it also supports quick and accurate decision making.

From that practical need, the Marketplace Transactional topic was deployed to simulate a transaction system of an e-commerce platform, and at the same time build an application to help users interact, manage data and analyze information effectively.

**2. Objectives of the topic**

The topic aims at the following main objectives:

Manage transaction information systematically:

The system allows handling of functions such as user login/registration, adding products, creating orders, managing categories and transaction details.

Transaction data analysis and statistics:

Use SQL queries to calculate revenue over time, total orders, best-seller rates, and analyze sales performance by user or category.

Visualize data with charts:

Integrate chart libraries (such as JFreeChart or Chart.js) to present data in the form of column, pie, line, area charts, etc. to help users grasp business trends more intuitively.

Building simulation application software:

Build a user interface using Java Swing (or other technology of choice such as web) with login, order management, product tracking and statistics dashboard access functionality.

**3. Structure and database**

Relational databases are designed with main tables such as:

users: Store user information including customers, sellers and administrators.

products: List of products being sold on the platform.

orders: Saves created orders, associated with users and order times.

order\_items: Specific item details in each order (quantity, price, product).

sellers: (optional) represents the sellers.

categories: Classify products by industry group.

Tables are linked via primary keys and foreign keys to ensure consistency, effectively supporting data querying and analysis.

**4. Main functions of the simulation system**

User registration/login with permissions (admin/staff/customer).

Search and filter products by name, type, price.

Manage shopping cart, create and pay for orders.

Manage product data, orders (for admin).

Query statistics over time: total revenue, best-selling products, number of orders.

Display analytical charts on dashboard: column, pie, line, bar, radar charts,...

**5. Meaning and practical application**

For study: Help students master the process of designing software systems according to the multi-layer model (MVC), processing real data, using object-oriented programming languages ​​(OOP), connecting to databases and visualizing results.

For practice: Can be expanded to apply in small sales systems, or used as a platform to develop real e-commerce applications.

For businesses: A miniature model to study customer behavior, optimize marketing activities, effectively manage supply chains and make business decisions.

**6. Expanded development direction**

Integrate online payment (Momo, ZaloPay, bank card).

Optimize user-friendly web/app interface.

Build a product recommendation system based on user behavior (machine learning).

Automatically analyze periodic sales reports and send emails.

**INTRODUCTION**

**1. Reason for choosing the topic**

In the context of e-commerce growing at an extraordinary speed, transaction data - especially from marketplace platforms such as Shopee, Lazada, Tiki, Amazon - not only has great economic value but is also an essential source of information for business analysis, performance evaluation and strategic planning. However, there is currently a lack of a centralized system that allows users to easily manage, retrieve and visualize transaction data by product, customer, time or order value. Having to analyze manually or rely on many separate tools makes it difficult for businesses to make timely decisions, while hindering the digital transformation process in operations.

Based on that reality, the team decided to implement the topic "Marketplace Transactional" with the goal of building a desktop application using Java language, supporting the management and analysis of e-commerce transaction data effectively and intuitively. The project applies the principles of object-oriented programming (OOP), exploits data structures in Java Collections Framework, connects to MySQL database to store and retrieve data, and integrates the JFreeChart library to visualize statistical charts related to revenue, number of orders and best-selling products. The application also ensures security through user authentication mechanism and reasonable authorization according to role (Admin, Staff).

Through this topic, the group not only hopes to provide a useful tool for store owners, analysts or users managing sales platforms, but also demonstrate the ability to apply learned knowledge to solve a practical problem, with high applicability and expansion in the field of software development for management and digital commerce.

**2. History of research on the problem**

### 2.1. General background

In recent years, e-commerce is no longer a trend but has become an indispensable part of modern economic and social life. Under the impact of the digital transformation wave and the development of technological infrastructure, consumers increasingly prefer online shopping, thereby promoting the emergence and strong development of intermediary sales platforms - also known as marketplaces.

Marketplaces such as Shopee, Lazada, Tiki, Sendo in Vietnam or Amazon, eBay, Alibaba in the world are playing the role of "electronic markets" connecting millions of sellers and buyers on a unified platform. From here, billions of transactions are made each year, generating a huge amount of rich and strategically valuable data. This data is not only the basis for processing orders, payments and shipping but also the foundation for conducting in-depth analysis of user behavior, consumer trends, managing sales performance and optimizing operations.

However, besides the outstanding successes of the e-commerce industry, there is still a big gap in managing and exploiting transaction data effectively, especially for small and medium enterprises (SMES), individual businesses, students and independent developers. That is also the reason for the birth of the Marketplace Transactional topic, aiming to research and build a software system that allows simulating the transaction process, managing and visualizing data effectively, easy to use and highly scalable.

### 2.2. The process of developing and researching the transaction management system

#### Phase 1: Traditional management –manualization with books and spreadsheets

Before the boom of e-commerce, most sales and transaction management activities were done manually. Shop owners and small businesses recorded orders, sales, and customer information on paper or basic spreadsheets such as Microsoft Excel. Although this method is simple and easy to implement, it has many serious limitations such as: low flexibility, no support for real-time statistics, prone to errors when entering data manually, and very difficult to expand when the number of orders increases.

#### Phase 2: Standalone POS Systems

As transaction volumes increase, many businesses have turned to standalone sales management software, commonly known as POS (Point of Sale). These software allow for inventory, invoice, customer and order management, support revenue reporting, best-selling product statistics, etc. However, they still lack in-depth data analysis features, cannot visualize consumption trends or performance over time, and maintenance and expansion are limited due to the lack of standardized software structure.

#### Phase 3: The emergence and strong development of e-commerce platforms

Since 2015, the emergence and rapid growth of e-commerce platforms such as Shopee, Lazada, Tiki, etc. in Vietnam have reshaped the entire shopping habits of consumers. Sellers are no longer limited geographically and can reach millions of customers nationwide with just one sales account. However, the ability to control and exploit transaction data on these platforms is still limited. Most of the reports provided by the system are in static format (CSV, Excel) and are designed for all sellers, without the ability to customize according to the individual needs of each user.

#### Phase 4: The Rise of Data Analytics and Information Visualization

As Big Data becomes a global trend, large enterprises are starting to integrate data analytics, artificial intelligence (AI) and data visualization tools to optimize operations and enhance user experience. Tools such as Tableau, Microsoft Power BI, Google Data Studio or Looker have helped thousands of organizations set up real-time analysis dashboards.

However, for small and medium-sized enterprises or individual businesses, accessing the above tools faces many barriers:

**High cost**(charge monthly or by number of users);

**Requires data analysis and SQL/BI programming skills**;

**Lack of direct integration with internal systems**;

**Data is fragmented and dependent on external systems**.

Therefore, the need for a simple, user-friendly, highly customizable transaction data analysis system that can run independently on the desktop is increasingly gaining attention.

### 2.3. Gaps in research and implementation practices

While there are many sales management systems and data analytics tools on the market, most of them are either:

**Design for large scale business**, difficult to deploy for small groups;

**Lack of personalization**for users;

**Does not support visual display of customizable charts based on flexible criteria**such as: monthly revenue, top selling products, order success/failure rate, analysis by customer group, etc.

Meanwhile, small groups such as IT students, online business beginners, or startup development teams often need:

A simple, lightweight system that runs offline if needed;

Easy to use interface, can assign user rights;

**Manage products, orders and users**effective;

Support data visualization with bar, pie, line charts;

Can be extended in the future to integrate more advanced features.

### 2.4. Conclusion

From the analysis of the development process and existing solutions on the market, it is clear that the Marketplace Transactional topic aims to fill: building a desktop software application (in Java), connecting to a MySQL database, applying object-oriented programming principles (OOP), organizing according to a multi-layer model (Model – DAO – Service – UI), integrating the JFreeChart charting library to visualize transaction data – thereby bringing practical value to end users.

This is not only a technical solution, but also a practical approach to the model of building software products according to systems thinking, meeting both academic goals and practical implementation capabilities in today's digital business life.

**3. Research purpose and tasks**

**3.1. Research purpose**

In the context of increasingly strong e-commerce development, marketplace systems (electronic markets) have become important platforms to help sellers and buyers connect, transact conveniently and quickly. Transaction data generated in these systems is a valuable resource that helps businesses grasp market trends, evaluate business performance and make strategic decisions.

Therefore, the purpose of the research topic "Analysis and development of Marketplace Transactional data management system" is:

Build a desktop application software system (Java Swing/JavaFX) capable of centrally and effectively managing e-commerce transaction data including information about products, customers, orders, suppliers and related transactions.

Analyze transaction data in multiple dimensions and criteria to create reports and visual charts, providing a comprehensive view of sales activities, helping administrators track revenue, order frequency, best-selling products, potential customers, and market trends.

Apply object-oriented programming principles and well-architected software design to ensure that the developed system is scalable, easy to maintain, reusable, while ensuring security and appropriate access permissions for different types of users in the marketplace.

Create an integrated technology platform between Java and MySQL, using the JFreeChart library to visualize data vividly and easily, while supporting CRUD operations (create, read, update, delete) data effectively and optimally for users.

Contribute to the digitalization and automation of sales management in small and medium-sized e-commerce businesses, helping them take full advantage of data to enhance business performance and improve customer experience.

Develop data-driven decision support tools that enable managers to predict market trends, evaluate marketing campaign effectiveness, and promptly adjust sales policies to optimize profits.

Enhance data analysis and statistical reporting capabilities, helping to reduce dependence on complex tools that require high expertise, thereby improving the work efficiency of management, accounting and business operations staff.

Ensuring stability, safety and flexibility during use, helping users to operate on multiple devices, expand the system according to future development needs, and easily integrate with other systems in the enterprise.

**3.2. Research tasks**

To achieve the above objectives, the topic focuses on implementing the following specific and detailed research tasks:

**3.2.1. Survey of current situation and analysis of requirements for transactional marketplace system**

Collect and research documents and reports on e-commerce activities and existing marketplace systems, thereby identifying transaction data characteristics and popular management models.

Analyze key marketplace transactional operations such as ordering, payment, shipping, returns, and product inventory management.

Survey the system usage needs of different user groups including administrators, sales staff, accounting department and internal customers.

Identify required functions and system evaluation criteria, including processing performance, scalability, data security, interface friendliness, and interoperability.

Establish business process modeling and initial data model as the basis for system design.

**3.2.2. System architecture design and data model**

Choose an appropriate architectural pattern (e.g. MVC or MVP) to clearly separate the interface, logic, and data access, to improve maintainability and extensibility.

Design an ERD that includes important tables such as products, customers, orders, order\_items, suppliers, and users.

Define data constraints, primary/foreign keys, indexes to ensure data integrity and optimize query performance.

Design user interfaces that respond to management, search, filter, and data entry operations in an intuitive and easy-to-use manner.

**3.2.3. Programming and developing desktop applications to manage and analyze transaction data**

Build a login module and assign user rights according to roles (Admin, Staff, Accountant) to ensure secure access.

Develop product management functions: add new, edit information, delete products and update inventory.

Build customer management functionality, allowing viewing of purchase history and classifying customers according to criteria such as purchase frequency and total purchase value.

Build an order management module with operations to add, edit, cancel orders, update order status and link order details.

Integrate advanced search features based on multiple criteria such as time period, product, customer, order status.

Develop reporting and charting systems using the JFreeChart library, including:

- Order rate by status

- Product ratio by category

- Average seller rating

- Total orders by day

- Number of orders by customer

- Number of sellers by rating range

- Number of products by category

- Number of orders per month

**3.2.4. System testing, evaluation and optimization**

Perform functional testing of the entire system, including various user action scenarios, security testing, and performance testing.

Evaluate user experience (UX) through feedback from test users to tweak interfaces and optimize workflows.

Optimize SQL queries, indexes, and data caches to speed up processing, ensuring fast application response even with large data.

Evaluate system scalability and maintainability, ensuring components are designed to be modular, easy to update and upgrade.

Ensure security measures such as password encryption, strict access authorization, and anti-SQL injection attacks.

**3.2.5. Documentation and final reporting**

Prepare detailed system user manuals for each user group.

Write a report summarizing the research, design, implementation and testing process, analyzing the results achieved compared to the set goals.

Prepare a presentation report, slides summarizing the research content and results.

**4. Research object and scope**

**4.1. Research subjects**

The research object of this topic is the e-commerce transaction system (Marketplace Transactional System) with a focus on transaction data arising from purchasing, payment and order management activities on the marketplace platform. Specifically, the topic focuses on researching and analyzing the following aspects:

Transaction data: Includes information about products, customers, sellers, orders, order details, payments, and order status.

Business process: From the process of customers searching, choosing products, ordering, paying to processing and managing orders by sellers and the system.

Data management system: Database and related components for storing, querying, updating transaction data in real time or on a session-by-session basis.

User Interface: Admin, customer and vendor screens for manipulating transaction data, displaying information, reporting, statistics and analysis.

Technology aspect: Applying Java programming technology, MySQL, and JFreeChart graphics library to build a desktop application capable of effectively managing and analyzing e-commerce transaction data.

**4.2. Scope of research**

The research scope of the topic is limited to the following frameworks and contents to ensure feasibility and focus on implementation:

Data Scope: The study is limited to real transaction data generated from marketplaces with key data tables such as customers, sellers, products, orders and related details. It does not extend to shipping, inventory or third party data.

Scope of functions: The topic focuses on building the following main functions:

Transaction Data Management and Processing (CRUD)

Query and analyze business indicators such as revenue, number of orders, best-selling products

Visualize data with charts and reports to support business decisions

User management, access rights in the system

Technology scope: Applying Java technology to develop desktop applications, using MySQL as a database management system, combining JFreeChart library to visualize data. No in-depth research on advanced security or web/mobile technologies.

Time horizon: Data and functionality are designed to be processed and analyzed within a certain time frame (e.g., transactions over a year or quarterly) to ensure performance and future scalability.

User scope: The system serves the main audiences including system administrators, business analysts and market sellers. It does not consider the end-user experience in placing orders directly through the platform.

**4. Research object and scope**

**4.1. Research subjects**

The research object of the topic is the system of managing and analyzing e-commerce transaction data in the marketplace environment, focusing on the following aspects:

Detailed transaction data: Includes all data related to the transaction process, from product information, customers, sellers, orders, order details, payment methods, order status to transaction history and events arising during the purchase and sale process.

Transactional business processes: Study the standard processes in marketplace operations such as ordering, order confirmation, payment, delivery, and processing of returns and cancellations. Understanding these steps helps build a system that accurately reflects actual operations.

System Architecture and Technology: Analyze how to design software systems using Java, MySQL, and supporting libraries such as JFreeChart for visualization, to ensure stable, efficient, and scalable operation.

User Interface and Experience: Focus on researching interface design for three main user groups: administrators, sellers and data analysts, ensuring a friendly interface, easy to use and optimized interactive experience.

**4.2. Scope of research**

The scope of the research is limited to focus on handling and developing core functions effectively, specifically as follows:

**4.2.1. Data scope**

Research and process data generated from the marketplace system including main data tables: customers, sellers, products, orders, order details, payment methods and transaction status.

Data focuses on completed and in-process sales transactions and does not include data related to warehousing, shipping, or marketing.

The research data period is limited to 1 to 3 years to ensure representativeness and feasibility of analysis.

**4.2.2. Functional scope**

Develop transaction data management functions such as creating, reading, updating and deleting (CRUD) transaction-related records on the marketplace.

Build query and data analysis features with important business indicators: revenue by day/month/year, number of orders, best-selling products, customer behavior analysis.

Integrate data visualization through various charts (column, pie, line) using the JFreeChart library to help users easily monitor and evaluate business situations.

Manage users with different access rights (administrator, analyst, seller), ensuring information security and confidentiality.

Do not develop complex functions such as online payment integration, delivery systems or multi-platform support (mobile, web).

**4.2.3. Scope of technology**

Use the Java programming language, the JavaFX desktop development platform, and the SceneBuilder interface support library to create software with a friendly, intuitive GUI.

Apply MySQL as a relational database management system to store, retrieve and process transaction data.

Incorporate the JFreeChart library for data visualization, helping to present business figures vividly and easily understood.

Focus on desktop application development, not web or mobile versions.

Apply object-oriented programming (OOP) model to build a system with flexible structure, easy to maintain and expand in the future.

**4.2.4. User scope**

The system is aimed at the following main users:

Administrator (Admin): Manage the entire system, including data, users and analytical reports.

Data Analyst: Use analytical tools to monitor and report on business performance, make data-driven forecasts and recommendations.

Seller: Manage your products and orders on the marketplace, track sales status.

It does not include end users who shop directly on marketplaces, nor does it study the purchasing experience.

**4.2.5. Scope of evaluation and testing**

Focus on evaluating the application's functionality in aspects such as accuracy, reliability, data processing performance, and visualization capabilities.

Tested on sample datasets from 1 to 3 years of trading to evaluate the ability to handle medium and large amounts of data.

Collect test user feedback to improve the interface and interactive experience.

No real testing conducted on real e-commerce marketplace environments or real user data.

**5. Research methods**

To ensure the scientific and effective implementation of the topic related to the e-commerce transaction system (Marketplace Transactional), the research team applies a system of diverse research methods, combining theoretical analysis, experimentation and modern software engineering. Specifically as follows:

**5.1. Data collection and processing methods**

Primary Data Source: Data is collected from the Marketplace Transactional dataset, which includes real transaction information or simulated data similar to popular e-commerce transactions. This dataset contains tables of data on orders, products, customers, sellers, payment information, and order status.

Collecting auxiliary data: In addition to the main data, the team also collected additional references, scientific articles, and case studies from leading marketplace platforms such as Amazon, Shopee, and Lazada to supplement knowledge and compare.

Data preprocessing: Data cleaning and normalization steps are very important to remove missing values, duplicate or incorrect data, ensuring high accuracy in analysis. Techniques used include: handling missing data (filling missing values ​​or removing), standardizing date formats, string formats, and normalizing numeric data.

Exploratory data analysis: Before performing in-depth analysis, the team conducts descriptive analysis to understand the structure, trends, and key characteristics of the data, helping to detect anomalies or important data patterns.

**5.2. System design and development methods**

Requirements analysis and synthesis: Based on the topic and reference documents, the team carefully analyzed the business requirements, user needs, characteristics and operating procedures of the marketplace system to build a list of functions that need to be developed.

System architecture design: Apply multi-layer architecture including layers: user interface (UI), business logic (Business Logic), and data access layer (Data Access). The clear separation of layers makes the system easy to maintain, expand and upgrade in the future.

Data Modeling: Design a relational database model, defining tables, relationships between tables, primary keys, foreign keys, and data constraints to ensure data integrity and optimize query performance.

Object-Oriented Programming (OOP) Development: Use OOP principles such as encapsulation, inheritance, polymorphism to design object classes that represent entities in the system (Users, Products, Orders, Transactions,...). This method helps the source code to be tightly organized, highly reusable and easy to maintain.

User Interface Development: Apply UI/UX design principles to build intuitive, user-friendly interfaces, including elements such as harmonious colors, clear layout, easy interaction, and optimized user experience.

**5.3. Software development methods and project management**

Software development process: The project is implemented according to the Waterfall Model with clear stages: analysis, design, implementation, testing, and maintenance. This model helps to control the progress and quality of the project well.

Programming and integration: Each functional component is developed independently and then integrated into the overall system. The integration process is carefully conducted to detect and correct errors promptly.

Using supporting tools: The team uses modern software development tools such as Java (JDK), MySQL for database, SceneBuilder for interface design, JFreeChart for data visualization, along with integrated development environments (IDEs) such as Eclipse or NetBeans.

Version Control: Use a source code control system (Git) to track source code change history, support team collaboration, and control code quality.

Descriptive statistical analysis: Apply basic statistical techniques to summarize and describe data characteristics such as mean, variance, frequency, data distribution, etc.

Advanced data querying: Design complex SQL statements to extract detailed information, analyze business trends such as monthly revenue, best-selling products, customer regions, etc.

Data visualization: Use the JFreeChart library to build a variety of charts (column, pie, line, area) to present analysis results in a visual, vivid way, helping users easily grasp important information.

Advanced analytics (if applicable): Can extend the application of data mining or machine learning techniques to predict trends, classify customers, or recommend suitable products.

**5.5. System quality testing and evaluation methods**

Unit Testing: Test each small functional module to ensure it works as designed.

Integration Testing: Tests the coordination between modules when integrated together.

System Testing: Testing the entire system in a near-real-world environment to detect errors or deficiencies.

Performance Testing: Evaluate the processing speed and load capacity of the system with large transaction volume, ensuring the system operates stably under real conditions.

Collect user feedback: Through surveys or interviews with test users, collect opinions to improve interface, features, and optimize experience.

Evaluation and refinement: Synthesize test results and feedback to refine the product, preparing it for long-term deployment and maintenance.

**5.6. Method of synthesizing and reporting results**

Research results synthesis: The results obtained from the analysis, design, development and testing processes will be systematically synthesized into scientific reports, technical reports and user manuals.

Presentation and reporting: Research results are presented in a written project report with a clear, scientific structure, accompanied by illustrative charts and tables to clarify important content.

**6. Project structure**

The project is organized into four main chapters, each chapter plays a separate role and is closely connected to each other to achieve the goal of researching, designing and implementing a Marketplace Transactional data analysis system effectively and scientifically. The overall structure helps readers easily follow and understand the steps, from basic theory to experimentation and product evaluation.

**• Chapter 1: Theoretical basis and research status**

Chapter 1 lays a solid theoretical foundation for the project. First, the chapter presents an overview of important knowledge related to the technology and methods applied in the system:

A detailed introduction to Object Oriented Programming (OOP), including principles such as Encapsulation, Inheritance, Polymorphism, and Abstraction. Understanding OOP is the foundation for designing flexible, maintainable, and extensible software.

Learn about JavaFX – a modern user interface technology that enables the creation of highly interactive, dynamic, and intuitive desktop applications.

Presents knowledge of MySQL database management system, data organization methods, query optimization and ensuring data integrity in transaction management systems.

Explains data structures in Java, such as linked lists, sets, maps, etc., that help in efficiently managing and processing data in application memory.

Introduce and analyze the JFreeChart library used to create visual charts for data analysis and reporting.

Survey and evaluate the tools and applications for analyzing sports and e-commerce data currently available on the market. Analyze the advantages and disadvantages of these tools, thereby pointing out the gaps and weaknesses that need to be overcome, as a basis for orienting the topic of product development suitable to reality and user needs.

**• Chapter 2: Solution and application design analysis**

This chapter focuses on building solutions and designing the overall system based on the requirements and objectives:

Propose a Marketplace Transactional system architecture solution, aiming for a scalable, easy-to-maintain, and secure solution.

Detailed analysis of functional requirements such as user management, transaction processing, reporting statistics, etc. as well as non-functional requirements such as performance, stability, scalability and security.

Using software analysis and design tools such as Use Case Diagram, helps illustrate key use cases, clarifying the roles and tasks of each type of user in the system.

Design an activity diagram to describe in detail the steps of implementing a business process, helping to ensure the operation process is accurate and logical.

Describe the MySQL database model in detail through the Entity-Relationship Diagram (ERD). This model illustrates the main entities, relationships, and necessary attributes, ensuring that data is logically organized and easy to query and update.

Describes data tables, primary keys, foreign keys, data integrity constraints, and indexes to improve access performance.

**• Chapter 3: System operation mechanism**

Chapter 3 focuses on the realization of the designed solution, detailing the operation of the software components in the application:

Introducing the classes and components of the MVC (Model-View-Controller) model:

**Model**: Represents data classes, representing entities in the system such as users, products, orders.

**KNIFE**(Data Access Object): Provides methods for accessing and manipulating data on the MySQL database.

**Service**: Process business logic, implement rules and procedures related to transactions and analysis.

**Controller**: Coordinates the flow of data between the interface and business layers, ensuring smooth user interactions.

Describes the user authentication and authorization mechanism, including the login process, password encryption using security algorithms, session management, and access authorization to ensure information security and confidentiality.

Presents detailed CRUD (Create, Read, Update, Delete) operations to manage data, allowing users to add, edit, delete and retrieve information about products, orders and customers flexibly.

Explains how to apply Java Collections Framework in storing, processing temporary data, supporting queries and calculations.

Detailed presentation on using JFreeChart library to create visual charts, helping users quickly grasp trends and trading situations in the system.

Refers to exception handling to improve system stability, and describes performance and memory optimization techniques for efficient system operation, especially when processing large amounts of data.

Evaluate system scalability and maintainability based on current design and operating mechanisms.

**• Chapter 4: Product testing and evaluation**

The final chapter focuses on testing and evaluating the system based on scientific and practical criteria:

Describe the experimental process in detail, including the experimental environment (computer configuration, software used), steps to test functions such as login, data management, report generation and chart display.

Build test scenarios for each major function, ensure the system operates as required and detect potential errors.

Analyze experimental results, evaluate criteria for data accuracy, operational stability, processing performance and user experience.

Presents the product's outstanding advantages such as user-friendly interface, fast data processing capability, high security and flexibility in management.

Frankly point out the limitations and weaknesses that still exist in the system, thereby proposing improvement directions to improve product quality in the next versions.

Compare the product with similar solutions already on the market, clarifying the improvements and competitive advantages that the topic brings.

Propose potential development directions such as integrating machine learning algorithms for predictive analysis, expanding the system to other areas, or upgrading the interface and user experience according to new technology trends.

**CONTENT SECTION**

**CHAPTER 1: THEORETICAL BASIS AND RESEARCH STATUS**

**1.1. Scientific basis**

**1.1.1. Object-oriented programming (OOP) and its role in e-commerce transaction systems:**

Object-Oriented Programming (OOP) is a modern software development paradigm, based on the thinking of modeling real-world entities as "objects" with characteristics (attributes) and behaviors (methods). OOP is not only a programming method, but also a software design philosophy, helping to increase the ability to extend, maintain and reuse source code.

In the context of building an e-commerce transaction system like the topic "Marketplace Transactional", OOP plays a core foundation role to help organize the entire system into specialized and tightly linked object classes. Each class represents a specific business element: User, Customer, Admin, Product, Order, Category, Transaction, Payment, etc. Applying OOP helps to clearly separate the processing logic of each part, from the interface to data retrieval, increasing the ability to maintain and expand in the future.

**a) Encapsulation:**

Encapsulation helps encapsulate data and data manipulation behavior in a single logical unit – an object. Data is protected from unauthorized access by making it private, and is only accessible through public getter and setter methods. In transaction systems, encapsulation helps protect sensitive information such as user passwords, payment card details, and orders from unauthorized access.

**b) Inheritance:**

Allows for building a class structure that inherits from each other, thereby reusing source code more effectively. In the system, the User class can be the parent class of the Customer, Staff, and Admin classes, helping to reduce code duplication and easily extend functionality. When new features need to be added (for example, account verification for Admin), just extend in the child class without affecting the parent class.

**c) Polymorphism:**

Allows objects to have the same interface but different implementations. For example, a processOrder() method can implement different logic depending on the type of object: Customer or Staff. Polymorphism gives the system flexibility in handling without having to distinguish between specific types of objects.

**d) Abstraction:**

Helps hide complex technical details and show only the necessary functions. For example:An IPaymentProcessor interface can define a processPayment() method, but each subclass like OnlinePayment, CODPayment will have a different implementation. This makes the system transparent, extensible and easy to control.

OOP not only increases the professionalism of the system but is also suitable for large-scale expansion, and is the optimal choice for modern e-commerce applications that require high security, performance and maintainability.

**1.1.2. JavaFX – A Platform for Building Modern User Interfaces**

**JavaFX**is an advanced user interface (UI) development platform for Java, designed to replace legacy technologies such as AWT and Swing. With its intuitive, flexible, and modern interface building capabilities, JavaFX is an ideal choice for highly interactive desktop applications such as e-commerce systems.

In this topic, JavaFX is used to build the entire interactive interface, including:

Registration form, login, product management, order list.

Dashboard reports with sales statistics charts and order analysis.

Smooth transition system between function screens using Transition effect.

**Outstanding advantages of JavaFX:**

**FXML:**Allows separation of interface (design in XML) and logic processing (in Java), helping frontend/backend programmers work independently.

**CSS Styling:**Customize the interface easily with web-like CSS files, giving it a professional and modern look.

**Animation & Effects:**Supports transition animations, hover effects, shadows, scales… to make the interface lively and user-friendly.

**Binding & Observables:**Allows 2-way data binding between UI and backend, suitable for real-time updating applications.

Compared to Swing, JavaFX provides a better interactive experience, richer graphics processing capabilities, and is a suitable choice for medium to large scale applications in the e-commerce field.

**1.1.3. MySQL Database – Efficiently manage transaction information**

Database is the backbone of any software system that stores and processes information. In the Marketplace system, MySQL – a popular relational database management system – is used to store all transaction data, customers, products, etc.

**Reasons to choose MySQL:**

Free, open source, high performance

Easy integration with Java via JDBC

Support SQL standard, flexible query

The database structure includes:

**users**: account management, user roles.

**products**: product information, price, quantity.

**orders, order\_items**: manage orders and product details.

**transactions**: record payments.

**categories**: classify products by group.

**reviews, favorites**: support advanced features such as rating, favorites.

**MySQL Highlights:**

Primary key, foreign key constraints: Ensure integrity between tables.

Index: Speed ​​up big data queries.

Data standardization: Minimize redundancy, optimize storage space.

**MySQL**is an ideal choice for medium and large systems, while ensuring scalability, security, and retrieval efficiency when combined with the DAO model in Java.

**1.1.4. Algorithms and data structures in business processing**

Behind the user interface and database, the system also needs to handle intelligent business logic, optimize performance and enhance the experience. Java provides a rich set of libraries supporting data structures and algorithms, helping to handle business tasks such as:

-Sort products by price, name, popularity

-Calculate total order value, tax, shipping fee

-Statistics of number of orders and revenue by day/month

-Data structures used:

**ArrayList, LinkedList**: dynamic product list processing

**HashMap:**used in charts or time-series data mapping

**HashSet:**remove duplicate customers or categories

**Queue:**order queue processing over time

**Comparator, Collections.sort():**Flexible product arrangement

Algorithms that can be implemented in the future:

**Binary Search:**Help find products quickly by code.

**Machine learning algorithms (Apriori, K-means):**Analyze purchasing behavior, personalize product recommendations.

The application of data structures and algorithms helps the system operate smoothly and respond quickly, especially when the number of transactions and users increases.

**1.1.5. JFreeChart and the role of data visualization**

Data visualization is an indispensable element in modern systems, helping managers easily identify trends and make accurate and timely business decisions. JFreeChart is a powerful chart library, supporting many types of charts with flexible customization capabilities.

Commonly used charts:

**Bar Chart:**Compare monthly/quarterly revenue

**Pie Chart:**Sales Proportion Analysis by Product Type

**Line Chart:**Track order trends

**XY Plot, Stacked Chart**: Analyze multiple variables at once

**Advantages of JFreeChart:**

Combine directly with data from MySQL

Customize color, font, size flexibly

Interactive support: zoom, highlight, hover information

Integrates well with Java Swing or JavaFX

Visualization is the key to turning raw data into valuable information, the "analytical eye" of the Marketplace system administrator.

**1.1.6. Security and user authentication in e-commerce systems**

In any transaction system, especially e-commerce, security is a vital factor. The system needs to ensure that user data is protected, unauthorized access is prevented, and user rights are clearly defined.

Security mechanisms applied:

Login with SHA-256 or bcrypt encrypted password

Authenticate users by role: Admin, Staff, Customer

Create a session and automatically log out when the session ends

User-level access permissions

Query database via PreparedStatement to prevent SQL Injection

The system complies with the 3-layer security principle: authentication - authorization - encryption, ensuring safety for both end users and business data.

**1.2. Current status and research orientation**

**1.2.1. Current status of e-commerce transaction system development**

E-commerce is witnessing remarkable growth in the digital era, especially the Marketplace model - which allows many sellers and buyers to interact and transact on a unified platform. "Giants" in this field such as Shopee, Lazada, Tiki, Amazon, Alibaba have shown outstanding business efficiency and global expansion. However, besides opportunities, there are also many challenges for researching and learning this model in the educational environment, especially at the student level.

Main difficulties when approaching modern e-commerce model:

High complexity of real systems

Platforms like Shopee or Amazon integrate hundreds of functions, from user management, orders, logistics, to marketing, livestream sales, product reviews... These functions not only require in-depth knowledge of programming, but also of system architecture, security, distributed databases, AI, Big Data.

Lack of transparency about source code and architecture

For competitive and security reasons, companies often do not make their system source code public. This makes it difficult for students and practitioners to access the internal structure, data flow, or actual operating mechanism to simulate or learn.

Cannot test or customize existing systems

Learning goes hand in hand with practice, but students cannot directly operate on the real system because they do not have access, cannot test business logic, create test errors, or improve functionality as desired.

Urgent need for a suitable simulation model

For the above reasons, building a medium-sized and open Marketplace e-commerce system, capable of simulating the main functions in an intuitive and easy-to-control way, is extremely necessary in the training environment:

Modeling the core functions of a modern Marketplace: User registration, product management, ordering, payments, order tracking, product reviews – ratings, and revenue statistics.

Suitable for student level: The system architecture is neatly designed, clearly divided into layers (UI - DAO - Model - Controller), using popular technologies such as Java, MySQL, Swing, helping students easily access, modify and upgrade.

Create a real-world testing and research environment: Allows testing of business logic, adding new features, analyzing order data, and even integrating modern technologies like charts (JFreeChart), chatbots, or AI in the future.

Orientation for complete product development: In addition to learning goals, the system can also be a sample platform for small startups, research groups or in-depth graduation theses on commercial transaction systems.

**1.2.2. Expansion orientation**

In the future, the system can be expanded in many potential directions, contributing to improving user experience, operational capabilities, and market competitiveness:

Multi-device Web/Mobile Platform Deployment

The system can be further developed as a web or mobile application, to increase user accessibility.

**Recommended technology:**

**Web:**Spring Boot (back-end) combined with ReactJS or Angular (front-end).

**Mobile:**Use Flutter or React Native to develop cross-platform (Android & iOS) apps.

**Benefit:**Increase flexibility, improve user experience, reach customers anytime, anywhere.

Integrate e-wallet and payment gateway

Integrate popular payment gateways such as VNPay, Momo, ZaloPay, PayPal to support fast and secure online payments.

**Additional features:**Save payment methods, refunds, one-click payments.

**Benefit:**Increase order conversion rate, enhance shopping experience.

Artificial Intelligence (AI) Applications

Personalized product recommendations based on purchase history and user behavior (recommendation system).

Fraud detection using machine learning models.

Smart customer care chatbot, 24/7 support, automatically answers frequently asked questions.

Technologies used: TensorFlow, Scikit-learn, GPT-based APIs.

Big Data Analytics

Deploy a system to collect and analyze real-time customer behavior data:

Tools: Apache Kafka (streaming), Apache Spark, Hadoop.

Application:

Customer segmentation

Predictive analytics

Optimize marketing campaigns (A/B testing, attribution tracking)

Migrating to Microservices Architecture

Split the system into small independent services such as: user management, orders, payments, products...

Supported technologies: Docker, Kubernetes, Spring Cloud.

Benefit: Easily expand and update each component without affecting the entire system, increasing load capacity.

DevOps – CI/CD Deployment

Apply Continuous Integration (CI) and Continuous Deployment (CD) processes to ensure code is always tested and updated continuously.

Tools: Jenkins, GitLab CI/CD, GitHub Actions, SonarQube.

Benefits: Speed ​​up software development, reduce errors, and shorten time to market.

Expanded third-party APIs

Allows connection to other systems such as CRM, ERP, delivery services (GHTK, GHN, Viettel Post), social networks (Facebook API, Zalo API).

Benefits: Automate business processes, support centralized management, expand multi-platform sales channels.

Enhanced system security

Apply encryption standards (AES, SSL/TLS), 2-factor authentication (2FA), detailed access authorization.

Periodically check for security vulnerabilities (vulnerability scanning) and upgrade attack prevention systems (firewall, WAF).

**Chapter 2: Proposed solutions and application design analysis**

1. **Proposed solution**

**2.1. System architecture overview**

The Marketplace Transactional system is built on a 3-tier architecture including:

**Presentation Layer (User Interface):**

The user interface is built using Java Swing, providing login and registration forms, customer and administrator dashboards, product and order management forms, and statistical charts. This layer is responsible for direct user interaction, displaying data, and accepting input requests.

**Business Logic Layer (Business Processing):**

This middle layer is responsible for handling business logic, including user authentication, handling CRUD (Create, Read, Update, Delete) requests on products, orders, users, checking access rights, calculating statistical data, processing payments, and managing order status.

**Data Access Layer:**

This layer connects and interacts directly with the MySQL database through DAOs (Data Access Objects) and corresponding model classes. This layer provides methods to query, update, delete and add new data in tables such as users, products, orders, order\_items,...

Separating the 3 layers makes the system easier to maintain, expand and upgrade later, and clearly separates the roles of each part in the application.

**2.2. Detailed design of components**

**2.2.1. Database Design — Expanding Details**

Database design is one of the most important steps in building a Transactional Marketplace system. A properly designed database not only helps to store data efficiently but also ensures integrity, scalability and fast retrieval as the system grows. Below is a detailed analysis of the main data tables, relationships between tables, and optimization techniques applied.

**Main tables:**

Users table

This is a table that stores system user information, including customers, sellers and administrators.

Main fields:

-id (INT, PRIMARY KEY, AUTO\_INCREMENT): unique identifier for each user.

-username (VARCHAR(50), UNIQUE, NOT NULL): user login name, required to be unique for authentication purposes.

-password\_hash (VARCHAR(255), NOT NULL): stores user password as hash (e.g. bcrypt), ensuring security.

-role (ENUM('admin', 'seller', 'buyer'), NOT NULL): classifies user roles to manage access.

-email (VARCHAR(100), UNIQUE): user email address, used for authentication or password recovery.

-created\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP): account creation time.

-updated\_at (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP): time the user information was updated.

Explain:

This panel helps manage system logins and permissions.

Using password\_hash instead of plaintext password increases security.

The role column helps distinguish user types to coordinate corresponding features in the system.

Products table

Store product information posted for sale on the marketplace.

Main fields:

-id (INT, PRIMARY KEY, AUTO\_INCREMENT): unique product code.

-name (VARCHAR(100), NOT NULL): product name.

-description (TEXT): detailed description of the product.

-price (DECIMAL(10,2), NOT NULL): product price.

-quantity\_in\_stock (INT, NOT NULL): current inventory quantity.

-seller\_id (INT, FOREIGN KEY): link to the seller (users table) responsible for the product.

-created\_at, updated\_at: product creation and update time.

Explain:

This table allows storing all the basic and necessary information of a product.

The seller\_id field helps identify who the seller is for management and statistics.

quantity\_in\_stock is important for proper inventory management and order processing.

Orders table

Manage orders created by users when purchasing products.

Main fields:

-id (INT, PRIMARY KEY, AUTO\_INCREMENT): order code.

-user\_id (INT, FOREIGN KEY): link to the user who placed the order (buyer).

-order\_date (DATETIME, NOT NULL): order time.

-total\_amount (DECIMAL(12,2), NOT NULL): total order value.

-status (ENUM('pending', 'processing', 'shipped', 'completed', 'cancelled'), NOT NULL): order status.

-created\_at, updated\_at: track creation and update time.

Explain:

This table stores general information about the order.

The status field helps manage the order processing, from placement to completion.

order\_items table

Save product details in each order.

Main fields:

-id (INT, PRIMARY KEY, AUTO\_INCREMENT): order detail code.

-order\_id (INT, FOREIGN KEY): links to the orders table.

-product\_id (INT, FOREIGN KEY): link to the product in the order.

-quantity (INT, NOT NULL): number of products in the order.

-unit\_price (DECIMAL(10,2), NOT NULL): price of each product at the time of order.

Explain:

This table helps separate product items in the order, creating flexibility in managing order details.

Unit prices are saved to avoid discrepancies if product prices change later.

Categories table (Optional)

If the system needs to classify products by category for easy searching and management.

Main fields:

-id (INT, PRIMARY KEY, AUTO\_INCREMENT): category code.

-name (VARCHAR(50), NOT NULL): category name.

-description (TEXT): detailed description of the category.

Explain:

Product categorization makes it easy for users to filter and search.

If a product belongs to multiple categories, an additional intermediate table product\_categories needs to be set up.

Logs table

Store system activity logs for monitoring and analysis purposes.

Main fields:

-id (INT, PRIMARY KEY, AUTO\_INCREMENT): log code.

-user\_id (INT, FOREIGN KEY): the person performing the action.

-action (VARCHAR(100)): describes the action (eg login, create order).

-description (TEXT): details of the action.

-timestamp (DATETIME): time the action occurred.

Explain:

Helps track activity history, supporting tracing in case of incidents or unusual behavior.

Relationships between tables

One-to-many relationship:

Each user can create multiple orders (users (1) - orders (n)).

Each order has multiple product items (orders (1) - order\_items (n)).

Each seller can manage multiple products (users (1) - products (n)).

Many-to-many relationship — if there are categories:

A product can belong to multiple categories, and a category can have multiple products.

Done using intermediate table product\_categories including product\_id, category\_id.

Indexing and optimization

Create indexes:

Primary Key: On the id columns of each table helps retrieve records by ID faster.

Secondary Indexes:

username in users table: supports quick user search when logging in.

product\_id and order\_id in order\_items table: helps to query order details quickly.

user\_id in orders table: query orders of a user.

seller\_id in products table: query products by seller.

Optimize data type:

Use appropriate data types and sizes to save memory and speed up queries.

For example, use VARCHAR(50) for username instead of VARCHAR(255).

Use DECIMAL(10,2) for product price to ensure decimal accuracy.

Query optimization:

Minimize queries, avoid SELECT \* and only get the necessary fields.

Use PreparedStatement for security and optimization.

Use pagination when querying large lists to avoid loading the entire data at once.

Data maintenance:

Set up a regular backup mechanism.

Periodically clean up old and unnecessary logs to avoid database bloat.

Scalability issues in database design

Scalability with increasing number of users and products:

Large data can be divided into multiple sub-tables (partitioning).

Consider using caching to reduce database query load.

Support for new features in the future:

Add a reviews panel for users to rate products.

Add wishlists table for wishlist.

The payment\_methods table stores payment method information.

Data normalization:

Design tables according to 3NF (Third Normal Form) to eliminate duplicate data and ensure consistency.

**2.2.2. Model Layer Design**

Each table in the database corresponds to a Model class in Java, for example:

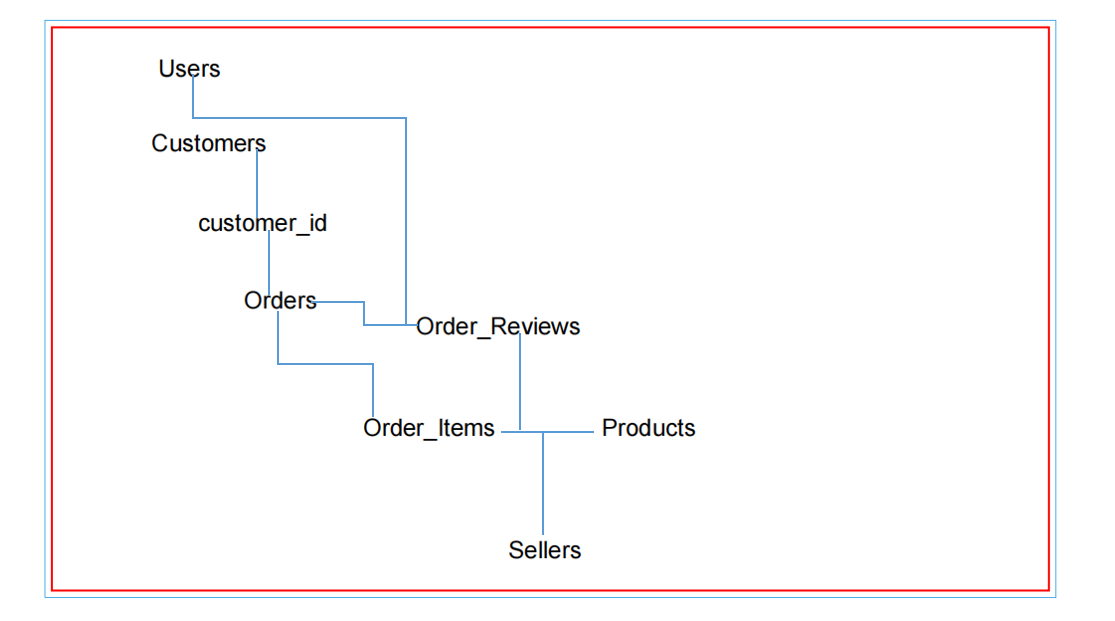
User.java with properties id, username, passwordHash, role, email,...

Product.java with properties id, name, description, price, quantityInStock, sellerId,...

Order.java includes id, userId, orderDate, totalAmount, status,...

OrderItem.java includes id, orderId, productId, quantity, unitPrice,...

Model classes only hold data, not containing complex processing logic.



*Figure 1.1 overview model*

**2.2.3. Designing the Data Access Layer (DAO Layer) — Expanding on the details**

The Data Access Object (DAO) layer is an essential component in the application architecture, responsible for intermediating between the Database layer and the Business Logic layer. DAO helps hide the technical details of querying and manipulating the database, thereby increasing the modularity, ease of maintenance and expansion of the application.

In the Marketplace Transactional system, DAO classes will be designed to interact directly with MySQL through the JDBC library, providing specific methods to retrieve, add, edit, and delete data according to each main table of the system.

**1. Role and goal of DAO Layer**

Separate data access logic from other layers:

Helps the Business Logic layer not to care about how data is stored and retrieved.

Encapsulate complex SQL operations:

SQL statements are processed centrally, making control and optimization easier.

Enhanced security:

Using PreparedStatement effectively prevents SQL Injection attacks.

Easy to expand and maintain:

When changing DB structure or access technique, just edit at DAO layer without affecting other layers.

Support for automated testing:

It is possible to mock DAO classes to test the Business Logic layer without accessing the actual DB.

**2. Design the main DAO classes**

Here is a detailed breakdown of the main DAO layers in the application:

***a) UserDAO.java***

Main functions:

**Manage user data in the users table.**

Typical methods:

getUserByUsername(String username): Finds and returns a User object by login name.

-addUser(User user): Add a new user to the database.

-updateUser(User user): Update user information, for example change password, email.

-deleteUser(int userId): Delete user by ID.

-getAllUsers(): Returns a list of all users.

Technical details:

Each method uses standard SQL statements, for example:

**SELECT \* FROM users WHERE username = ?**

PreparedStatement is used to pass parameters, preventing injection.

The query results are mapped to a User object (model).

***b) ProductDAO.java***

Main functions:

**Manage products in the products table.**

Typical methods:

getAllProducts(): Get a list of all products for sale.

getProductById(int productId): Get product details by ID.

addProduct(Product product): Add new product to inventory.

updateProduct(Product product): Update product information.

deleteProduct(int productId): Delete product.

getProductsBySellerId(int sellerId): Get products by specific seller.

Technical details:

Queries like SELECT \* FROM products WHERE id = ?.

Support pagination, filtering, sorting in list retrieval methods.

***c) OrderDAO.java***

Main functions:

**Manage orders in the orders table.**

Typical methods:

createOrder(Order order): Create a new order.

getOrdersByUserId(int userId): Get a list of orders by user.

updateOrderStatus(int orderId, String status): Update order status.

getOrderById(int orderId): Get order details.

Technical details:

Use transactions when creating orders to ensure consistency between the orders and order\_items tables.

Optimized query to retrieve orders and related information (e.g. join order\_items table).

***d) OrderItemDAO.java***

Main functions:

**Manage items in each order, order\_items table.**

Typical methods:

addOrderItem(OrderItem orderItem): Add new item to order.

getItemsByOrderId(int orderId): Get all products in the order.

updateOrderItem(OrderItem orderItem): Update product quantity or price.

deleteOrderItemsByOrderId(int orderId): Delete all items in the order (when canceling the order).

Technical details:

Query using safe parameters.

Integrate with OrderDAO to process transactions.

**3. Important design and engineering principles applied**

Using PreparedStatement

Every query uses PreparedStatement to avoid SQL Injection errors.

Helps increase performance by preparing query statements once and reusing them multiple times with different parameters.

Returns Model objects

The query results are mapped to the corresponding Model classes (User, Product, Order, OrderItem).

Helps the Business Logic layer work directly with understandable objects, without having to deal with SQL ResultSets.

Connection and resource management

The DAO is responsible for properly closing the connection, ResultSet, and PreparedStatement after execution is complete to avoid resource leaks.

Connection pool can be used to increase DB access performance.

Exception Handling

DAOs should catch and handle SQL errors, and can wrap them in custom exceptions for easy handling by upper layers.

This helps to clearly distinguish between data access errors and business errors.

Transaction Management

When manipulating multiple related tables such as creating orders and order items, it is necessary to ensure data integrity by using transactions.

When part of an operation fails, the entire operation is rolled back to avoid out-of-sync data.

Query optimization

Write precise queries, avoid retrieving redundant data.

Caching techniques can be used for static queries like product catalog, to reduce DB load.

1. **DAO layer model in the system**

+----------------+ +-----------------+ +----------------+

| UserDAO | | ProductDAO | | OrderDAO |

+----------------+ +-----------------+ +----------------+

| +getUserBy... | | +getAllProducts | | +createOrder |

| +addUser() | | +addProduct() | | +getOrdersByUser|

| +updateUser() | | +updateProduct()| | +updateStatus()|

| +deleteUser() | | +deleteProduct()| | +getOrderById()|

+----------------+ +-----------------+ +----------------+

+--------------------+

| OrderItemDAO |

+--------------------+

| +addOrderItem() |

| +getItemsByOrderId()|

| +updateOrderItem() |

| +deleteItemsByOrder()|

+--------------------+

Each DAO class is independent, dedicated to one table, following the Single Responsibility principle.

The Business Logic layer will call DAO methods to perform data related operations.

**5. Benefits of applying DAO Layer**

Increased scalability and reusability:

When extending the system, simply add or modify DAO classes without changing the entire source code.

Easy maintenance:

Clear separation between layers makes debugging and upgrading easier.

Increased security:

Using PreparedStatement and tightly managing DB operations reduces the risk of security flaws.

Multi-platform support:

DAO can be adapted to work with other database management systems if needed.

**2.2.4. Service Layer Design**

The Service layer combines DAOs to perform complex operations, for example:

Login authentication: check if username exists, compare encrypted password.

Ordering: check inventory, reduce product quantity, create orders and order items in a transaction.

Product management: check user rights, validate input data.

Statistics: calculate total revenue over time, number of products sold,...

The service layer also handles logic checks, error messages, and business processes.

**2.2.5. User Interface Design (UI Layer) — Expanding Details**

The user interface (UI Layer) is the part that directly contacts the user, playing an important role in creating a smooth, intuitive and easy-to-use experience. In the Marketplace Transactional system, the interface is developed on the Java Swing platform, a standard library for building desktop applications with rich UI components such as JFrame, JPanel, JTable, JButton, JTextField,...

**1. Interface architecture and screen organization**

The application is designed according to a clear layered model, the UI Layer is the top layer, responsible for:

Display data retrieved from the Business Logic layer.

Collect and process user interactions.

Send interaction requests back to the Business Logic and DAO layers to process the business.

The interface is divided into main modules corresponding to user and administrative functions:

***a) Login/Registration form***

Is the starting point of the application.

Allows users to enter username and password to log in.

Has the function of registering a new account for users who do not have one yet.

The interface has basic components such as JTextField for data entry, JPasswordField for password, JButton buttons for login and registration.

Have a clear error response mechanism (e.g. "Incorrect username or password").

***b) Customer Dashboard***

Main screen for customers after successful login.

Display the product list as a paginated table (JTable) or list (JList).

Has the function of searching and filtering products according to criteria such as name, price, category.

Allow customers to select products and add them to cart with desired quantity.

Show shopping cart with options to edit quantity or delete products.

Provide payment button, call order processing.

Use intuitive controls like JButton for actions, JComboBox for category filtering, JTextField for searching.

Friendly interface design, harmonious colors, easy to see to enhance user experience.

***c) Administration Dashboard***

For system administrators or staff.

There are many subsystems such as:

User management: add, edit, delete accounts.

Product management: add new, edit information, update inventory.

Order management: view, approve, cancel orders.

Revenue statistics: display revenue charts, number of orders over time using JFreeChart library.

The interface includes detailed data tables (JTable), input forms (JPanel with multiple data fields).

Functions are organized into tabs or navigation menus for easy access.

***d) Product detail form***

Display product details when a customer or administrator selects a specific product.

Information fields include name, description, price, quantity in stock, image (if available).

Allow customers to select the quantity they want to buy and add to cart.

Allows administrators to edit or update product information.

***e) Order processing form***

Display order details for both customers and administrators.

Allows administrators to change order status (processing, delivered, canceled).

Allows customers to view order history and current order status.

The interface has tables listing the items in the order, total amount, order date, and payment method.

**2. Interface design principles**

***a) User-friendly and intuitive***

Use a reasonable layout, easy to observe, avoid confusion.

Buttons and input fields are located for easy access.

The colors are harmonious and contrast enough for users to read easily.

Error messages and clear input instructions.

***b) Fast, smooth response***

The interface responds instantly to user actions.

Buttons and panels have hover and click effects to increase interactivity.

Use multi-threading to avoid interface “freezing” when performing data queries.

***c) Ensure consistency***

The screens have a uniform design in style, color, and font.

The function buttons are always located in the same position for easy remembering by users.

***d) Extensibility and maintainability***

The interface is organized in a component-based model, making it easy to add new or modify.

Clear division between interface processing logic and business processing logic.

**3. Technologies and libraries used**

***Java Swing:***Provides standard UI components, powerful support for desktop applications.

***JFreeChart:***Chart library helps build revenue charts, visual statistics for admin dashboards.

***JDBC:***Connect and manipulate MySQL database, data is retrieved to display in tables and charts.

Multi-threading: Use SwingWorker or other multi-threading techniques to handle long tasks (like loading data) without freezing the UI.

**4. User Experience (UX)**

Easy login and registration:

Support new users to quickly create accounts.

Secure password information with server-side encryption.

Search and browse products efficiently:

Smart search bar with keyword suggestions.

Filter products by category and price range to help customers find them faster.

Convenient shopping cart:

Allows easy editing of product quantities.

Display total amount, apply promotion (if any).

Fast and secure payment:

Confirmation notice before completing order.

Clearly display order status after payment.

Effective management for admin:

Data tables are easy to read, paginated and sorted.

Visual charts help you quickly grasp the business situation.

**5. Potential improvements in UI Layer**

Add responsive interface: even though it is desktop, it can flexibly adjust its size to fit many screens.

Integrate real-time notifications: use push or polling techniques to update order status, instant messages.

Multi-language support: helps users easily switch between languages.

Enhance accessibility: design UI for users with special needs such as the visually impaired, adjust fonts and colors for readability.

**2.3. Main business processes**

**2.3.1. Login and authorization**

User enters username and password.

The system checks if the username exists, compares it to the encrypted password (bcrypt).

Authorization by role: admin, seller, buyer.

After successful login, go to the respective dashboard.

**2.3.2. Product Management — Expand Details**

In the Transactional Marketplace system, product management is one of the core and very important functions, directly affecting business operations and user experience. Seller and admin roles are allowed to perform product management operations including adding, modifying and deleting products.

Add new product

The seller or administrator will fill in all product information in the form, including:

Product name (required, cannot be blank, cannot be too long).

Detailed description (full recommended, helps customers understand the product clearly).

Selling price (must be a positive, reasonable number).

Inventory quantity (must be a non-negative integer).

Product catalog (if any).

Product photo (if supported).

The system strictly validates the input data before allowing it to be saved to the database:

Check that the product name does not duplicate existing products.

Check for valid price and quantity formats.

Required fields cannot be left blank.

Once the data is valid, the product information is saved to the products table with the corresponding fields.

The user receives a success or error message if any.

Product Modifications

Allow seller or admin to select products already in the list, change necessary information.

Edit operations must also perform the same data check as when adding new ones.

The system will update the information in the database, and also save the previous version information to support recovery or checking when needed.

Delete product

When a user chooses to delete a product, the system performs the following checks:

Check if that product exists in any completed or pending orders.

If the product has been sold (is in the order\_items table with an order status of "shipped" or "processing"), the system will not allow deletion to ensure order history integrity.

If the product has never been sold, the system allows deletion, completely deleting the record in the products table.

This check helps avoid loss of transaction history data or falsification of business reports.

**2.3.3. Ordering and Payment — Expand Details**

The ordering and payment function is an important step that determines the success of a transaction in the system.

Ordering process

Select product and quantity:

Customers select products from the product list displayed on the dashboard.

Enter the quantity you want to buy, the system checks that this quantity does not exceed the current inventory quantity.

If the quantity exceeds the stock, an error message is displayed and a re-entry is requested.

Add to cart:

The product and quantity are added to the customer's shopping cart.

Shopping cart can edit quantity or delete products before proceeding to checkout.

Pay:

When the customer presses the payment button, the system performs the following steps:

Check inventory: Make sure the quantity of products in the shopping cart is still sufficient in stock.

Create new transaction: Create a new order in the orders table with user information, order date, order status (default is “processing”).

Create order items: Create detail records in order\_items corresponding to each product and quantity in the cart.

Update Inventory: Decreases the product inventory quantity corresponding to the ordered quantity.

Use transactions in the database to ensure all the above steps are performed in sync, avoiding shortages or excess products when an error occurs in the middle of the operation.

Confirmation notice:

Once completed, the system sends a successful order confirmation message to the customer.

Notifications can be displayed directly on the interface or sent via email (if integrated).

Update order status so customers and admins can track.

Handling special situations

If inventory is insufficient during checkout, the system will cancel or request the customer to reduce the purchase quantity.

Handle network or system errors that occur mid-way by rolling back transactions to keep data consistent.

**2.3.4. Statistics and reports — Expand details**

System administrators can access reporting and statistical functions to evaluate business situations and make timely decisions.

Main types of reports

Revenue over time:

Revenue reports are analyzed by day, month, quarter, year.

Help administrators grasp sales trends by stage.

Number of products sold:

Statistics on the quantity of each type of product sold in a specified period of time.

Analyze best-selling products or products with long inventory.

Order by status:

Statistics on the number of orders being processed, delivered, and canceled, helping to monitor operational efficiency.

Total revenue and profit:

Calculate total revenue, gross profit by period.

Data visualization tools

Use the JFreeChart library to create column, line, pie, area charts...

The chart is dynamically updated based on actual data from the database.

The reporting interface provides filters, for example:

Select time period.

Filter by product type.

Filter by order status.

Allows exporting reports as PDF or Excel files (if integrated).

**2.4. Security Risk and Solution Analysis — Expand Details**

**2.4.1. Data security**

Encrypt user passwords:

Use bcrypt algorithm to encrypt password before saving to database.

The bcrypt algorithm is resistant to brute force attacks due to its ability to slow down the hashing process.

Prevent SQL Injection attacks:

All SQL queries use PreparedStatement with safely bound parameters.

Do not insert user data directly into SQL statements.

Input Validation:

Restrict and carefully check all user input to avoid malicious code injection (XSS).

Use libraries or special character filtering methods in text fields.

Encrypt sensitive data:

In addition to passwords, important information such as session tokens and payment information are also encrypted or strictly protected.

**2.4.2. Access security**

Strict decentralization:

Clear authorization system according to roles (admin, seller, customer).

Each role is only allowed access to appropriate functions, for example:

admin: full system management rights.

seller: manage your products.

customer: purchase, view order history.

Functional access restrictions:

Check access at the backend layer, avoid unauthorized access via API or interface.

Save access and operation history:

Record login and logout logs.

Record important data changes for auditing and investigation when needed.

Use HTTPS (if deploying web service):

Protect transmitted data from theft or tampering.

**2.4.3. Data protection in case of errors**

Using transaction:

Ensure data integrity in complex operations (ordering, payments, inventory updates).

If any step fails, the entire transaction will be rolled back to the previous state.

Error checking and exception handling:

Handle errors that arise (exceptions, database connection errors) appropriately.

User friendly and clear error message feedback interface.

Regular data backup:

Plan to backup your data regularly to avoid loss in the event of hardware failure or system failure.

System monitoring:

Set up automatic alerts when unusual errors or signs of intrusion are detected.

**2.5. Detailed test plan**

**2.5.1. Functional Testing — Expanding Details**

Functional testing is an important step to ensure that the system components operate according to business requirements and have no serious errors before being put into actual operation. In this project, testing is performed synchronously from the data layer (DAO), service layer (Service) to user interface (UI).

**1. Unit Test**

Target:

Make sure each method and function in DAO and Service classes works correctly with valid inputs and handles exceptions well.

Scope:

Methods to query, add, edit, and delete data in a DAO (e.g. ProductDAO.addProduct(), OrderDAO.createOrder()).

Business processing methods in Service (eg: ProductService.validateProduct(), OrderService.processOrder()).

How to do:

Use automated testing frameworks like JUnit for Java.

Create test cases including:

In case of valid data (happy path).

In case of incorrect or missing data (to test exception handling).

Boundary data case.

Use Mock Objects to mock relevant components if needed (e.g. mock database connection).

Expected results:

Methods return correct results, update data correctly.

Exceptions are caught and handled properly.

**2. Integration Test**

Target:

Test the interoperability of modules when working together.

Scope:

Test key business flows such as:

Login and authorization.

Add/edit/delete products and update lists.

Order, check inventory, pay.

How to do:

Set up a test environment with a test database (with sample data).

Execute complex business scenarios, from start to finish.

Check data consistency after operations.

**3. UI Testing**

Target:

Ensure that interface elements display correctly, buttons and input fields function properly and respond promptly.

Scope:

Main screens such as:

Login/Register form.

Customer dashboard (view products, search, add to cart, checkout).

Administration dashboard (manage users, products, orders, statistical charts).

Product detail form, order processing.

How to do:

Perform each function manually and check the returned results.

Test correct/incorrect input cases.

Test error responses, warnings, and notifications on the interface.

Use tools that support automated UI testing (if possible) like Selenium for web applications or equivalent frameworks for desktop applications.

Usability Testing:

Evaluate the user-friendliness and ease of use of the interface.

Test response speed, user effects.

**4. Functional security testing**

Check access authorization cases:

Users who do not have permission to access the feature are denied access.

Users can only view, not edit, or delete data without sufficient permissions.

Test vulnerable points like multiple failed logins, dangerous input data.

**5. Basic performance testing**

Evaluate the system's ability to handle multiple users simultaneously (especially ordering and inventory updates).

Check response time of critical functions.

**6. Record and handle errors**

All errors detected during testing are recorded in detail:

Describe the error, steps to reproduce, screenshots if needed.

Troubleshoot and retest to ensure the bug is completely fixed.

**2.5.2. Performance and Load Testing — Expanded Details**

Performance and load testing is an important step to evaluate the system's ability to operate stably and smoothly when many users access it simultaneously and process large amounts of data. This helps detect and fix bottlenecks, ensuring an uninterrupted user experience.

**1. Load Testing**

Target:

Evaluate the system's load capacity when the number of users accessing at the same time increases.

Method:

Use a Virtual Users simulation tool like JMeter, Gatling, or similar.

Set up real-world access scenarios like logging in, browsing products, adding to cart, placing orders, viewing reports.

Gradually increase the number of concurrent users to test the load limit.

Measurement index:

Average response time of operations.

Error rate increases as load increases.

Resource usage (CPU, RAM, network).

Expected results:

System response time is below acceptable levels (e.g. < 3 seconds for key operations).

No timeout or crash when loading below the specified maximum.

**2. Performance Testing**

Target:

Evaluates the speed of processing key tasks and system optimization capabilities.

Method:

Measure processing time for important operations: login, download product list, create order, update inventory, generate statistical reports.

Analyze SQL queries using tools like MySQL EXPLAIN, slow query log to identify slow queries.

Optimize indexes, queries, and data table structures.

Expected results:

Fast response time, no interruption to the experience.

Queries are optimized, minimizing latency and resource consumption.

**3. Stress Testing**

Target:

Determine the system's maximum load limit and react when the limit is exceeded.

Method:

Increase the load beyond normal to check if the system hangs or crashes.

Evaluates recovery capability after load reduction.

Expected results:

The system does not crash and can recover quickly when loading back to normal.

**2.5.3. Security Testing — Expanded Details**

Security testing aims to detect potential vulnerabilities in the system, ensuring user data and system resources are protected against real attacks.

**1. Test change SQL Injection attack**

Target:

Ensures that the system is not exploited through malicious SQL queries.

Method:

Try to insert malicious SQL statements into input fields (login, search, add product forms).

Check system and database responses for changes, data exposure, or errors.

Evaluate the use of Prepared Statements throughout the DAO.

Expected results:

The system rejects or safely processes inputs that show signs of attack.

No data was exposed or changed abnormally.

**2. Cross-Site Scripting (XSS) Attack Testing**

Target:

Prevent JavaScript or malicious code from being inserted into input fields.

Method:

Try entering script snippets into data-receiving fields (e.g. comments, product names).

Check the display on the interface, protect data and prevent malicious code from executing.

Expected results:

Malicious code is not stored or executed on the user's browser.

Input data is tightly controlled (escape, sanitize).

**3. Test for unauthorized access**

Target:

Ensure only authorized users can access sensitive functionality or data.

Method:

Try accessing admin pages or APIs with a regular user account.

Attempt to perform unauthorized operations such as deleting products or editing orders without permission.

Expected results:

The system denies access and clearly states.

Do not expose sensitive data.

**4. Evaluate the ability to protect user data**

Target:

Ensure personal data, passwords and sensitive information are kept secure.

Method:

Check password encryption (e.g. bcrypt).

Evaluate the storage of sensitive data according to security standards.

Test data recovery capabilities in the event of a crash.

Expected results:

Passwords and sensitive data are not stored in plain text.

The system ensures data integrity and security.

**2.6. Implementation and operation plan**

**2.6.1. Implementation objectives**

The Marketplace Transactional system implementation and operation plan is built with the goal of ensuring that the system's actual operation is effective and controlled, while ensuring that the system is always stable, safe and meets the needs of user groups including: customers, employees and administrators.

In addition, the system needs to be easy to maintain, upgrade and expand in the future.

**2.6.2. Detailed implementation phase**

***Phase 1: Prepare development environment and data***

Setting up development tools:

IDE: Eclipse or IntelliJ.

Language: Java (using Swing to build interface).

Database: MySQL, managed via SQLyog.

Supported libraries: JFreeChart for charts, JDBC for database connection.

Create database:

Create marketplace database and tables: users, products, orders, order\_items, customers, sellers.

Import sample data for development and testing.

***Phase 2: Develop core functionality***

Building platform functions:

Decentralized login/registration interface.

Main screens: Customer Dashboard, Admin Dashboard.

CRUD functions for products, orders, users.

Search, filter, sort data in the display table.

Integrated statistical charts:

Statistics on revenue, best-selling products, number of orders per month,...

Use column, pie, line, and summary charts.

Applying object-oriented programming principles:

Organize the system according to a multi-layer model: model, dao, ui.

Increased scalability, maintainability and code reuse.

***Phase 3: System Testing***

Functional Testing:

Make sure everything works as it should: from login to order processing.

Performance Testing:

Measure the time to query large data from the database.

Check the speed of displaying statistical charts.

Security Testing:

Check role-based access restrictions.

Check sensitive data (encrypt passwords, prevent SQL Injection).

***Phase 4: Internal testing deployment***

Install the system on some local machine (localhost).

Invite test users to experience the system.

Record feedback on:

User Interface (UI/UX).

Features work.

Stability and processing speed.

Make edits based on real-world feedback.

***Phase 5: Official Deployment***

System deployment:

Put the application into production on the server (or admin computer).

Set up a real database, add real product data and users.

**Instructions for use:**

Write user documentation for each type of user.

Organize internal training sessions as needed.

Final check:

Make sure all features work properly and no errors occur.

**2.6.3. Operation, monitoring and maintenance**

System Operation

Regular activity monitoring:

Monitor system logs: user actions, orders, errors.

Monitor system performance: response time, query performance.

Assign system administrator:

A technical person is assigned to handle the problem promptly.

Maintenance and upgrades

Regularly back up data:

Automatically backup your database daily or weekly.

Store backups in multiple locations to reduce the risk of data loss.

Version update:

Regular software updates to fix bugs and improve performance.

Upgrade features according to user needs.

Feedback and improvements:

Collect feedback from users via forms or interviews.

Synthesize and evaluate to include in the new version upgrade plan.

**2.6.4. Conclusion**

Implementation and operation planning is an indispensable part of the system development process. Reasonable phase division, careful preparation, combined with real testing and evaluation will help the Marketplace Transactional system operate effectively, meet business goals and user experience.

**2.7. User interface design analysis**

**2.7.1. Interface design goals**

The GUI design aims to create a friendly, easy-to-use and professional system that allows users to interact effectively with the system. The interface must meet the following criteria:

User-friendly.

Easy to navigate, fast to operate.

Clearly show user functions and roles.

Unify layout, color and design style.

**2.7.2. System user analysis**

The system divides users into 3 main groups:

| **User Group** | **Role in the system** | **Interface needs** |
| --- | --- | --- |
| Client | Search, order | Simple, intuitive interface, quick product operation |
| Staff | Order and product management | Clear layout, easy to filter and search data |
| Administrator | Manage the entire system | Comprehensive interface, with charts and statistics |

**2.7.3. Applied design principles**

KISS principle (Keep It Simple, Stupid): the interface should be minimalistic but fully functional.

Consistency: use consistent style, color, and font across screens.

Instant feedback: each user action has clear feedback (success/failure notification).

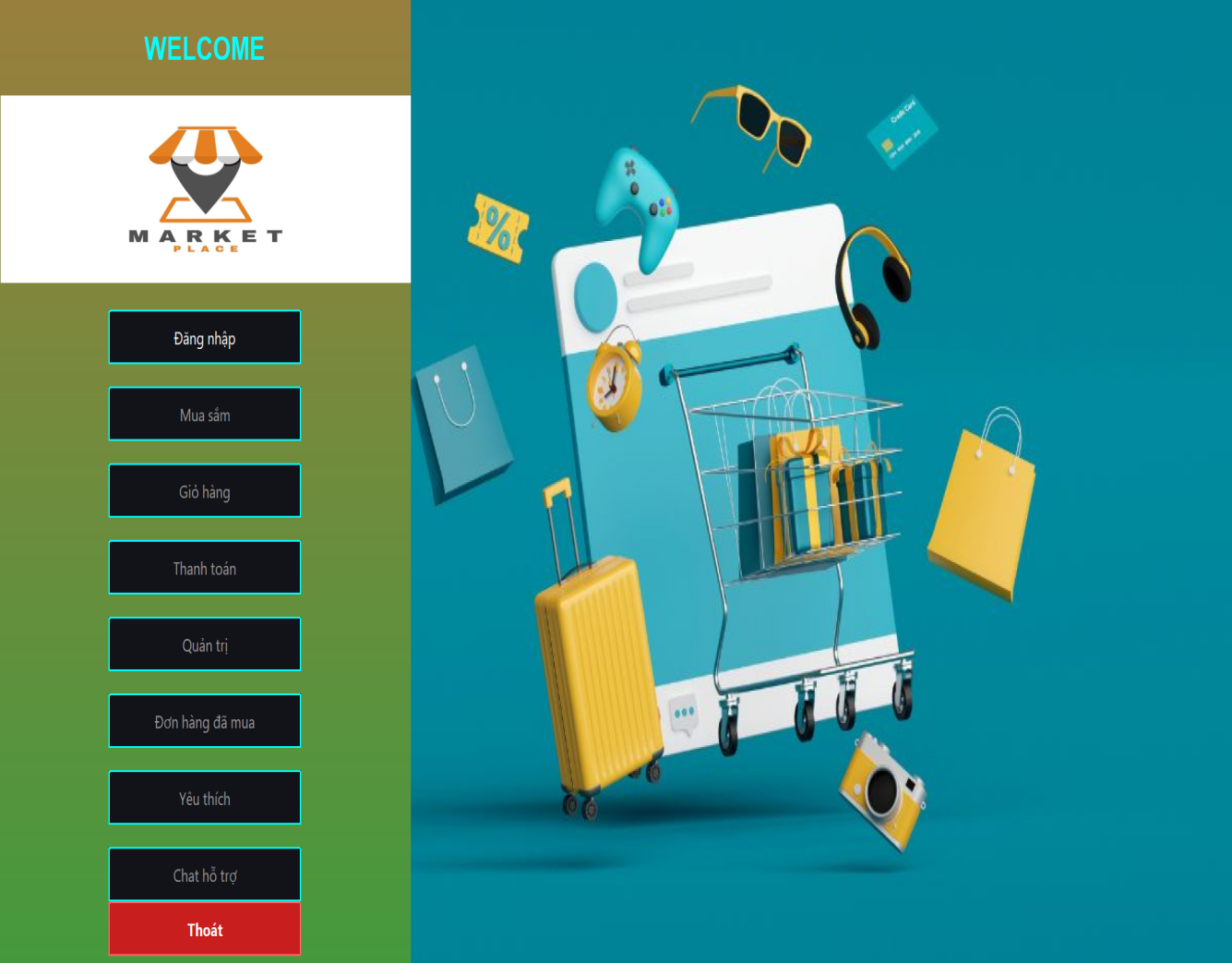
Visibility of system status: always displays system status and operation results.

Accessibility: easy-to-see colors, no eye strain, easy mouse and keyboard operations.

**2.7.4. Overall user interface structure**

**2.7.4.1 Overview of Home page interface layout**

The Home page interface is the first component that users come into contact with when starting the application. This is not simply a working window but also an important starting point that helps orient and navigate all user activities in the system. Therefore, the design of the Home page interface needs to ensure aesthetics, convenience in use and consistency in the organization of interface components.



*Figure 1.2 main home screen*

**1. Overall interface structure**

The interface is divided into two main parts horizontally:

The left part is the navigation Sidebar:

Has a width of about 30% of the total screen width, occupies the entire height of the window, and is used to place the application's main function buttons.

The right side is the main content area:

Occupying the remaining 70% of the screen area, it is used to display background images or sub-windows that are opened according to each function selected by the user.

**2. Navigation Sidebar**

The sidebar is very important, as it acts as the main dashboard for the user:

Color design and motion effects:

The sidebar is designed with a continuous gradient color transition effect based on the change of hue in the HSB color system. This effect not only brings dynamic beauty to the interface but also creates a lively, modern feeling, helping users feel comfortable when interacting with the application for a long time.

Logo and welcome message:

At the top of the sidebar, the application logo and the words “WELCOME” are placed solemnly in bold Times New Roman font, large font size, bright cyan color. This creates friendliness, professionalism and increases brand recognition right from the first time opening the application.

List of main navigation buttons:

The function buttons are arranged vertically in a straight line, with reasonable distance, creating convenience when pressing on the touch screen or with the mouse. Each button has a corresponding illustrative icon to help increase the ability to quickly recognize the function.

Login based function button state:

By default, the “Login” and “Logout” buttons are disabled when the user is not logged in. This is a smart access control measure to ensure that only authenticated users are allowed to operate sensitive features such as shopping, payment, or system administration. When successfully logged in, these buttons will be enabled, making the user experience seamless.

Hover interaction effect:

As the user hovers over each button, the button's background color changes to a lighter or darker tone, providing instant visual feedback, helping the user recognize which button is being pointed to, thereby minimizing operational errors.

Prominent “Exit” button:

The “Exit” button is designed with a bright red color when hovering to clearly warn that this is an important and irreversible function, helping users avoid accidentally pressing it, causing data loss or sudden exit of the application.

**3. Main content area on the right**

The main content display is optimized to ensure aesthetics and easy flexibility to change according to each function:

Sharp and aesthetic background images:

The high-resolution background image covers the entire panel, creating a visual highlight for the interface. The dark background color is chosen to harmonize with the colorful sidebar, creating visual balance and reducing eye strain when working for long periods of time.

Ability to open sub-windows for each function:

When the user selects a function on the sidebar, a corresponding child window (independent JFrame window) will be opened in the center of the screen. This design helps to keep the main interface layout intact while creating a specialized working environment for each feature, helping users focus and easily manage the tasks they are performing.

Future expandability:

This area is flexibly designed, easy to expand or change the displayed content depending on future development requirements such as adding dashboards, charts, data entry forms... without having to change the entire interface structure.

**4. Features that support user experience**

Clear visual feedback:

Changing button colors and states based on user actions (selecting, hovering) increases interaction and reduces confusion during operations.

Access status control:

Users are required to log in before performing major operations, ensuring data security and safety.

Intuitive, easy-to-use interface:

All interface components are scientifically arranged according to user habits, helping to operate quickly and minimizing learning time.

Meaning and Impact of Home Interface Design

Designing the Home page interface is not just about arranging buttons and images, but also includes many psychological and technical factors to enhance user experience and efficiency:

Create trust and attraction:

A beautiful, modern and professional interface helps increase users' trust in the quality of the software, and at the same time attracts them to use it for a long time.

Clearly orient users:

The navigation sidebar helps users easily identify available functions, minimizing getting lost or time-consuming feature searches.

Optimize work performance:

Separate child windows open in the center, allowing users to focus on each specific task without being distracted by other interface elements.

Data protection and access:

Disabling function buttons until successful login is a basic but necessary layer of security, preventing unauthorized access or manipulation errors.

Analysis of the performance of the main components

Login Button:

When enabled, opens a login window for the user to enter account information. The system authenticates the information, if successful, activates other function buttons and updates the status showing the username or access rights.

Shop Button:

Allows access to product list and performs product selection, filtering, and search operations.

Cart Button:

Display details of selected products, allowing to modify quantity or delete products.

Payment Button:

Open the payment form with payment methods, summarize the order and confirm the transaction.

Admin Button:

Granted to admin or high-level staff, providing comprehensive management functions such as product, order, and customer management.

Purchased Orders button:

Displays user's purchase history, allowing to check order status and details.

Favorite Button:

Allows users to view a list of products marked as favorites for easy repurchase or later viewing.

Chat button support:

Connect directly with customer support to get your questions answered or problems resolved quickly.

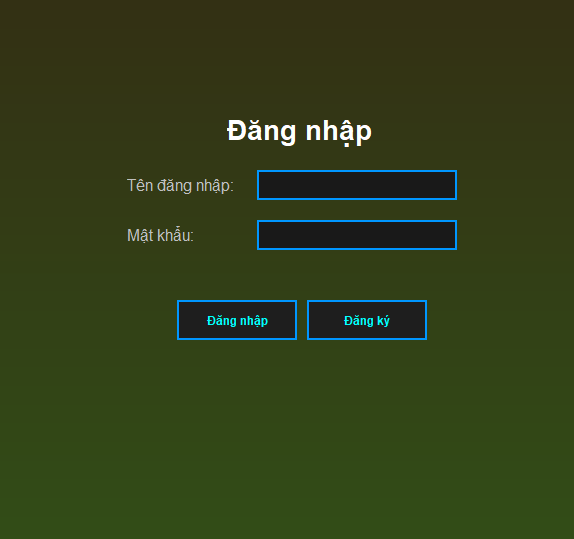
Exit Button:

Close the application completely, making sure to free up resources and end the session.

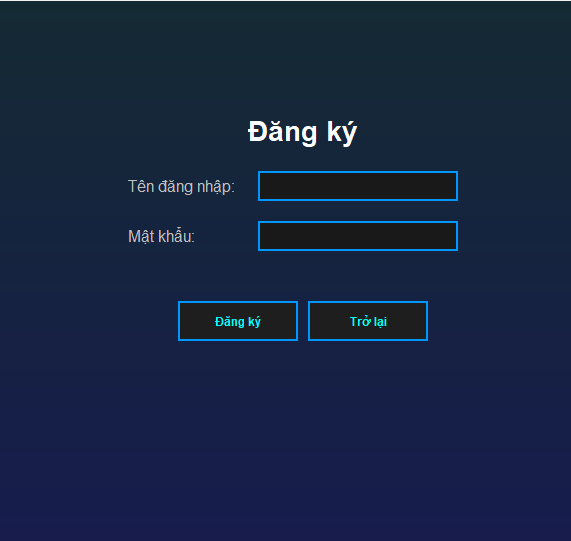
**2.7.4.2 – Login and Registration Screen**

**1. Functional objectives**

The Login and Registration screens are the starting point of the entire system. They allow users to authenticate their identity and assign permissions based on their roles. Keeping these two screens separate increases security, eases maintenance, and ensures a better user experience.



*Figure 1.3 login screen*



*Figure 1.4 registration screen*

**2. Login Screen (LoginForm.java)**

User Interface

Designed with Java Swing, clean layout, easy to manipulate, can be extended to add background, logo or animation.

Includes ingredients:

Login Name (JTextField)

Password (JPasswordField)

Login Button

Register button to open RegisterForm

Processing function

When the user clicks Login:

The system reads data from the input fields.

Call the authenticate method in UserDAO.

If the account is valid:

The system takes the user role.

Open the corresponding interface:

AdminDashboard (administration)

CustomerDashboard (customer)

SellerDashboard (seller)

If invalid: display error message.

3. Registration Screen (RegisterForm.java)

User Interface

Built with GridBagLayout for clean and scalable layouts.

Ingredients include:

Username

Password

Full name

Select roles: admin, seller, customer via JRadioButton

Sign Up Button

Processing function

When you click Register:

The system checks whether the fields are empty or not.

Create a User object with the input information.

Call UserDAO.addUser() to add to the database.

If registration is successful:

Show message and close window.

If failed (duplicate name or DB error):

Show warning.

**4. Link to the overall system**

| Related Components | Role |
| --- | --- |
| **UserDAO** | Perform login testing and add new users to the database. |
| **Database (table)**users) | Save user information, including username, password, full name, role. |
| **AdminDashboard** | Opens after logging in as admin, allowing system-wide management. |
| **CustomerDashboard** | Main interface for customers to view products and make purchases. |
| **SellerDashboard** | Interface for sales staff, order and product processing. |

**Functional linkage diagram**

flowchart TD

A[LoginForm] -->|login successful| B[Check user role]

B -->|admin| C[AdminDashboard]

B -->|customer| D[CustomerDashboard]

B -->|seller| E[SellerDashboard]

A -->|click register| F[RegisterForm]

F -->|Registration successful| A

Streaming

- Log in

User → Enter name & password → Click "Login"

↓

LoginForm class → calls UserDAO.checkLogin()

↓

Database table `users` → return result

↓

→ If correct → get role → open corresponding dashboard

→ If false → display error

- Register

User → Enter name, password, full name, select role → Click "Register"

↓

RegisterForm class → call UserDAO.addUser()

↓

Check if login name already exists in `users` table

↓

→ If not yet → Add to DB → Success message

→ If already exists or DB error → Show error

**2.7.4.3 – Shopping Function (CustomerDashboard.java)**

**1. Objective**

The shopping feature is a central component of the sales system for users who act as customers. The main goals of this feature are:

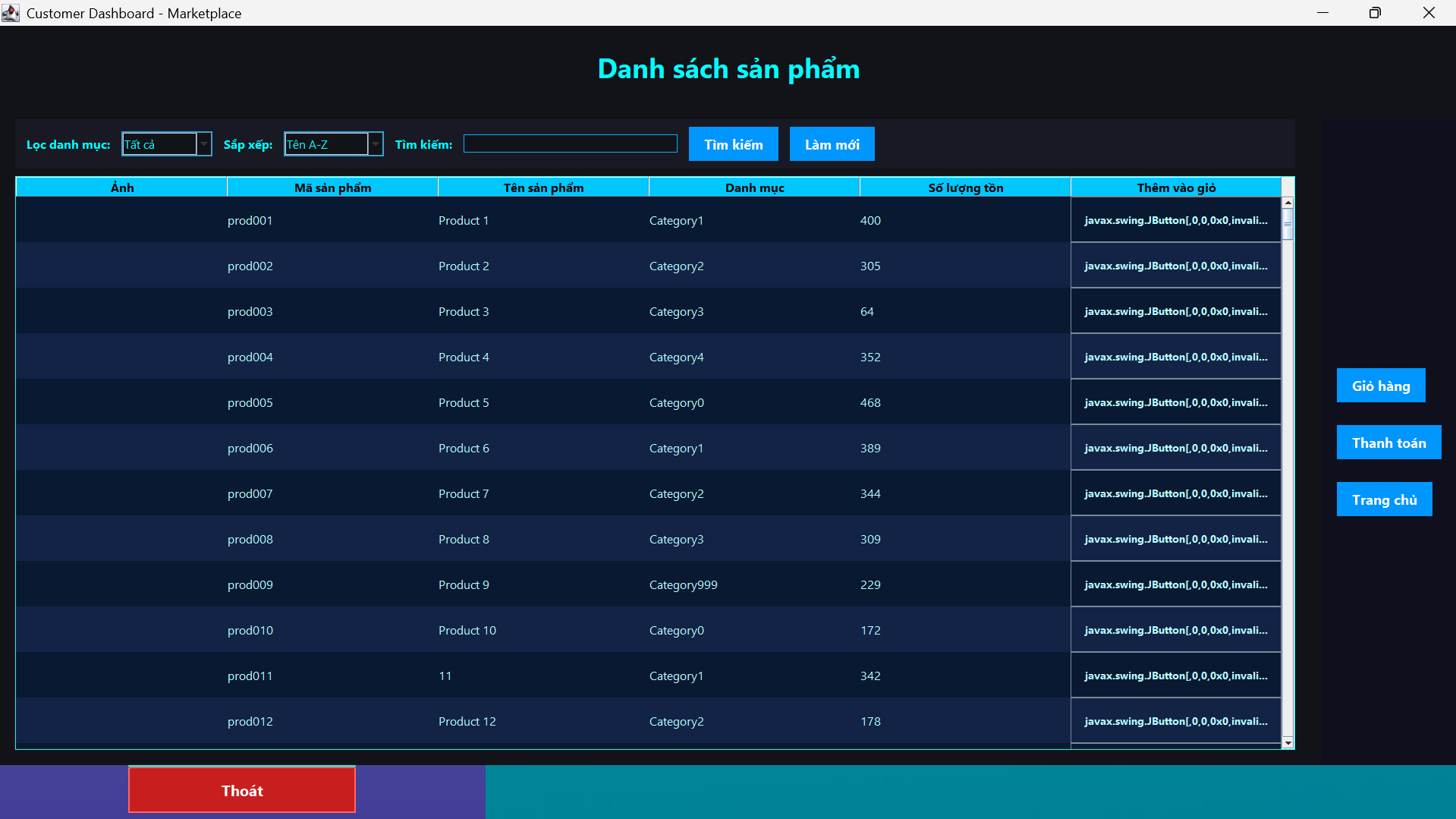
Display a list of all products for sale.

Allows users to browse products by category or search by name.

Provides flexible sorting by criteria such as product name or inventory quantity.

Allows users to view product details, add products to cart, and pay for orders in an intuitive and convenient way.

This function acts as an intermediary between the database that stores product information and the end-user's consumption behavior. It clearly demonstrates the layered architecture principle in software design with a user interface (UI) layer, a business logic layer, and a data access layer (DAO).



*Figure 1.5 product list*

**2. User interface overview**

CustomerDashboard is built on the Java Swing library, inherits JFrame, and is divided into the following main elements:

**2.1 Title**

Display interface name: "Product List" with large JLabel, creating a professional feel.

Large, bold, easy to recognize font.

**2.2 Product filters**

Includes navigational elements for filtering, searching, and sorting:

JComboBox cboCategoryFilter: filter by product category. Category data is taken directly from the products table or categories table.

JComboBox cboSort: sort products by criteria:

Name A–Z (ascending).

Name Z–A (descending).

The number increases.

Quantity decreased.

JTextField txtSearch: search by free keyword (product name).

JButton btnSearch, btnReset: perform a search or refresh the interface.

**2.3 Product table (JTable tblProducts)**

Display product list with columns:

Product image (ImageIcon).

Product code.

Product name.

Category.

Inventory quantity.

"Add to cart" button.

Each row corresponds to a Product object, mapped from the database.

Use DefaultTableModel and custom TableCellRenderer to display images and buttons in the table.

**2.4 Right control panel**

JButton btnCart: opens the shopping cart window (CartView).

JButton btnCheckout: proceed to checkout.

JButton btnHome: return to home page or log out.

**3. Source code structure and business processing**

**3.1 Get product list from database**

Through the ProductDAO class:

List<Product> products = productDAO.getAllProducts();

The ProductDAO class uses JDBC to retrieve data from the products table in the marketplace database.

The data is then passed to loadProducts() to display on the JTable.

**3.2 Filtering and searching**

When the user selects a filter or enters a keyword:

String keyword = txtSearch.getText().trim();

String selectedCategory = cboCategoryFilter.getSelectedItem().toString();

String sortOption = cboSort.getSelectedItem().toString();

The system filters the productList using appropriate conditional expressions.

Then call loadProducts(filteredList) again to update the interface.

**3.3 Product arrangement**

Use Collections.sort() or Comparator to sort the product list based on:

product.getName() → product name.

product.getQuantity() → quantity in stock.

**3.4 Add products to cart**

Each product line has an "Add to Cart" button.

When clicked, the product is added to the HashMap<String, Product> cart.

Processing code:

Product selectedProduct = productList.get(rowIndex);

cart.put(product.getId(), product);

**3.5 Display cart**

Open CartView (which is a JDialog) to display all products in the cart.

Users can update quantity, delete products or proceed to checkout.

**3.6 Order Payment**

Processing in checkout() function:

Browse cart, create Order and corresponding OrderItem.

Add to orders and order\_items tables.

Subtract the inventory quantity of each product.

Clear cart when finished.

**4. Interaction between layers**

ProductDAO: handles all product retrieval logic.

Product: product model class.

CartView: sub-interface to view and update shopping cart.

OrderDAO, OrderItemDAO: process when making payment.

**5. Techniques applied**

OOP (Object Oriented):

Define classes Product, Order, Cart, Customer, ProductDAO, CartView with encapsulation and separation of duties principles.

Advanced Java Swing UI:

Customize JTable to display images and buttons.

Combines TableCellRenderer and TableCellEditor.

MySQL and JDBC:

Communicate with the marketplace database via DBConnection.

Data Caching:

The product list is saved as List<Product> for quick local manipulation.

Event Handling:

Handles ActionListener, MouseAdapter, ComboBoxSelection, KeyListener events.

**2.7.4.4 – Cart function (CartView.java)**

**1. Objective**

The shopping cart function is an important component in the sales system, serving users in managing selected products before proceeding to payment. The main goals of this function include:

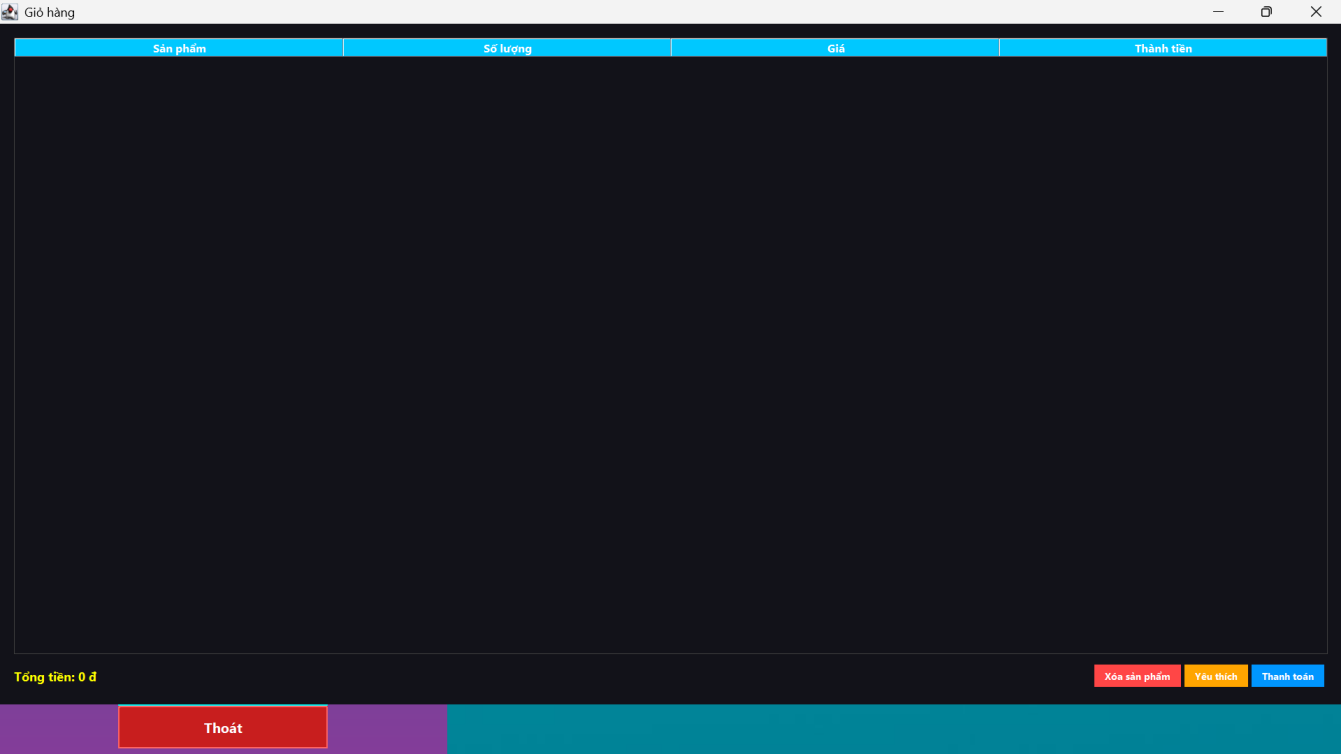
Displays a list of products added to the cart, along with the quantity and price of each product.

Allows users to edit, delete products from the shopping cart.

Support adding products to favorites list (Favorites).

Provides intuitive and convenient order payment features.

This function clearly demonstrates the layered architecture principle in software design with a user interface (UI) layer, a business logic layer, and a shopping cart data management layer.



*Figure 1.6 shopping cart screen*

**2. User interface overview**

CartView is built on the Java Swing library, inherits from JDialog or JFrame, and is divided into the following main parts:

**2.1 Title**

Display a "Your Cart" label or similar using JLabel in a large, bold, easily recognizable font.

**2.2 Shopping cart table (JTable tblCart)**

Display list of products in cart with columns:

Product Name

Quantity

Price

Total (Quantity × Price)

Use DefaultTableModel to manage table data. The table supports selecting multiple rows for manipulation.

**2.3 Function buttons**

btnRemove: Removes the selected item from the shopping cart. Allows you to delete multiple items at once.

btnFavorite: Add the selected product to the favorites list. If the product already exists, notify the user.

btnCheckout: Initiates order checkout when cart is not empty. Shows confirmation dialog.

btnClose: Close the shopping cart window, return to the previous screen.

**2.4 Total information**

Display the total amount of all products in the cart using a JLabel that is automatically updated whenever there is a change in the cart.

**3. Source code structure and business processing**

**3.1 Storing shopping cart data**

Shopping cart data is managed in Map<String, Product> or HashMap<String, Product>, where the key is the product code (productId), the value is a Product object with quantity.

**3.2 Display data on the board**

The loadCart() method will transfer data from the Map to the table's DefaultTableModel to display the product name, quantity, price and total amount for each row.

**3.3 Remove products from cart**

When the user selects a product and presses the "Delete" button, the corresponding products will be removed from the Map and the table and total will be updated.

**3.4 Add product to wishlist**

When "Favorite" is pressed, the system checks if the product does not already exist in the favorites list (FavoriteManager or similar), adds it to this list and notifies success, or if it already exists, warns the user.

**3.5 Order Payment**

The checkout() function will:

Check cart is not empty.

Create Order and OrderItem objects corresponding to the products in the cart.

Record orders and order details into the database via OrderDAO and OrderItemDAO.

Update product inventory quantity.

Clear current cart after successful checkout.

Notify the user of the results.

**4. Interaction between layers**

Product: product data model.

CartView: interface and shopping cart processing.

FavoriteManager: manage favorites list.

OrderDAO, OrderItemDAO: handle writing orders to the database.

DBConnection: connects to a MySQL database.

**5. Techniques applied**

OOP (Object Oriented):

Apply the principles of encapsulation, layering and separation of responsibilities between classes.

Advanced Java Swing UI:

Use JTable, DefaultTableModel to manage table data; use ActionListener for handling button events.

Local data processing:

Use Map to manage shopping cart for quick operation and easy interface updates.

Database interaction via JDBC:

Communicate with MySQL to store and retrieve order and inventory data.

Event Handling:

Handle user events like adding, removing products, checkout, and switching interfaces.

UI Effects:

Create grid gradient effects to enhance user experience.

**2.7.4.5 – Order Checkout Function (CheckoutView.java)**

**1. Objective**

The Checkout function is the final step in the shopping process, helping users confirm orders, apply promotions, calculate total payments and record orders into the system. The main goals include:

Display order details (product, quantity, unit price, total).

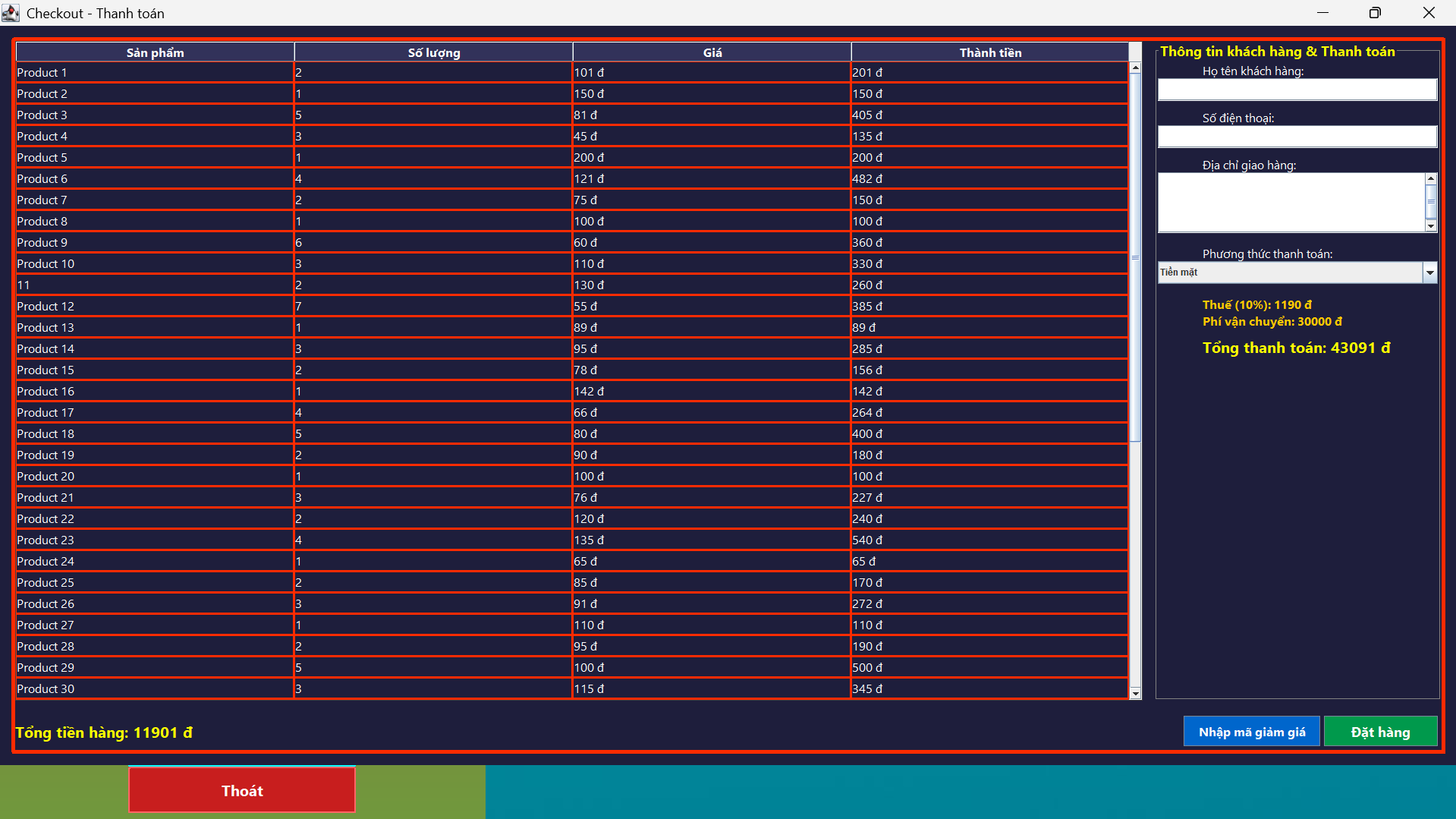
Support entering discount codes, recalculating total payment.

Provide tax and shipping information.

Allows payment confirmation and saving orders to the database.

Clear and friendly payment results notification.

This functionality also clearly demonstrates the layered architecture: User Interface (UI), Business Logic, and Data Management (DAO).



*Figure 1.7 payment screen*

**2. User interface overview**

The CheckoutView interface is built using Java Swing, inheriting from JDialog, including the main components:

**2.1 Title**

Display the text “Order Confirmation” using JLabel, large, easy to recognize font.

**2.2 Order table (JTable tableOrderDetails)**

Display a list of products on order with columns:

Product Name

Quantity

Unit price

Total (quantity × unit price)

Data is displayed via DefaultTableModel.

**2.3 Financial information**

lblTotal – Total price of goods (excluding taxes/fees)

lblTax – Tax 10%

lblShippingFee – Fixed Shipping Fee

lblFinalTotal – Final payment total

Values ​​are calculated and updated automatically based on product list and coupon code.

**2.4 Function buttons**

btnApplyDiscount: Enter discount code (currently supports code “GIAM10” for 10,000 VND discount)

btnConfirmOrder: Make payment, write data to database

btnCancel: Cancel the transaction and close the window

**3. Source code structure and business processing**

**3.1 Order data storage**

Product data in the order is transmitted as a List<Object[]>, each element includes:

Product Name (String)

Quantity (int)

Price (double)

Sum up into money and save temporarily for calculation.

**3.2 Displaying table data**

Method setOrderDetails(List<Object[]> items):

Fill the data into the tableOrderDetails.

Calculate the total cost of goods (subtotal).

Call updateTotals() to calculate additional taxes, fees, and final payment total.

**3.3 Apply discount code**

When the user enters a valid code (e.g. "GIVEAWAY10"), the system:

Subtract the corresponding amount from the total order.

Display success or error message.

**3.4 Calculating total cost**

Totals(double subtotal) update method:

Tax 10% on total goods price.

Fixed shipping fee (10,000 VND).

Recalculate total payment = (subtotal - discount) + tax + fee.

**3.5 Order Payment**

The confirmOrder() method does:

Check if the order list is valid.

Create new order (Order) and order details (OrderItem).

Write data to database via OrderDAO, OrderItemDAO.

Clear cart or update product status.

Show success message.

**4. Interaction between layers**

| **Ingredient** | **Function description** |
| --- | --- |
| Product | Product model, including name, price, code, quantity. |
| CheckoutView | Order confirmation and processing interface. |
| OrderDAO | Save orders to MySQL (orders table). |
| OrderItemDAO | Save each product in the order to MySQL (order\_items). |
| DBConnection | Connect to database (using JDBC). |

**5. Techniques applied**

OOP (Object Oriented):

Apply the principles of encapsulation, layering and separation of responsibilities between classes.

Advanced Java Swing UI:

Use JTable, DefaultTableModel to manage table data; use ActionListener for handling button events.

Local data processing:

Use Map to manage shopping cart for quick operation and easy interface updates.

Database interaction via JDBC:

Communicate with MySQL to store and retrieve order and inventory data.

Event Handling:

Handle user events like adding, removing products, checkout, and switching interfaces.

UI Effects:

Create grid gradient effects to enhance user experience.

**2.7.4.6 – System Administration Function (AdminDashboardView.java)**

1. Objective

The system administration function is a central control point for administrators or high-level staff, helping them monitor and operate sales activities. The main objectives include:

Product management: add, edit, delete products.

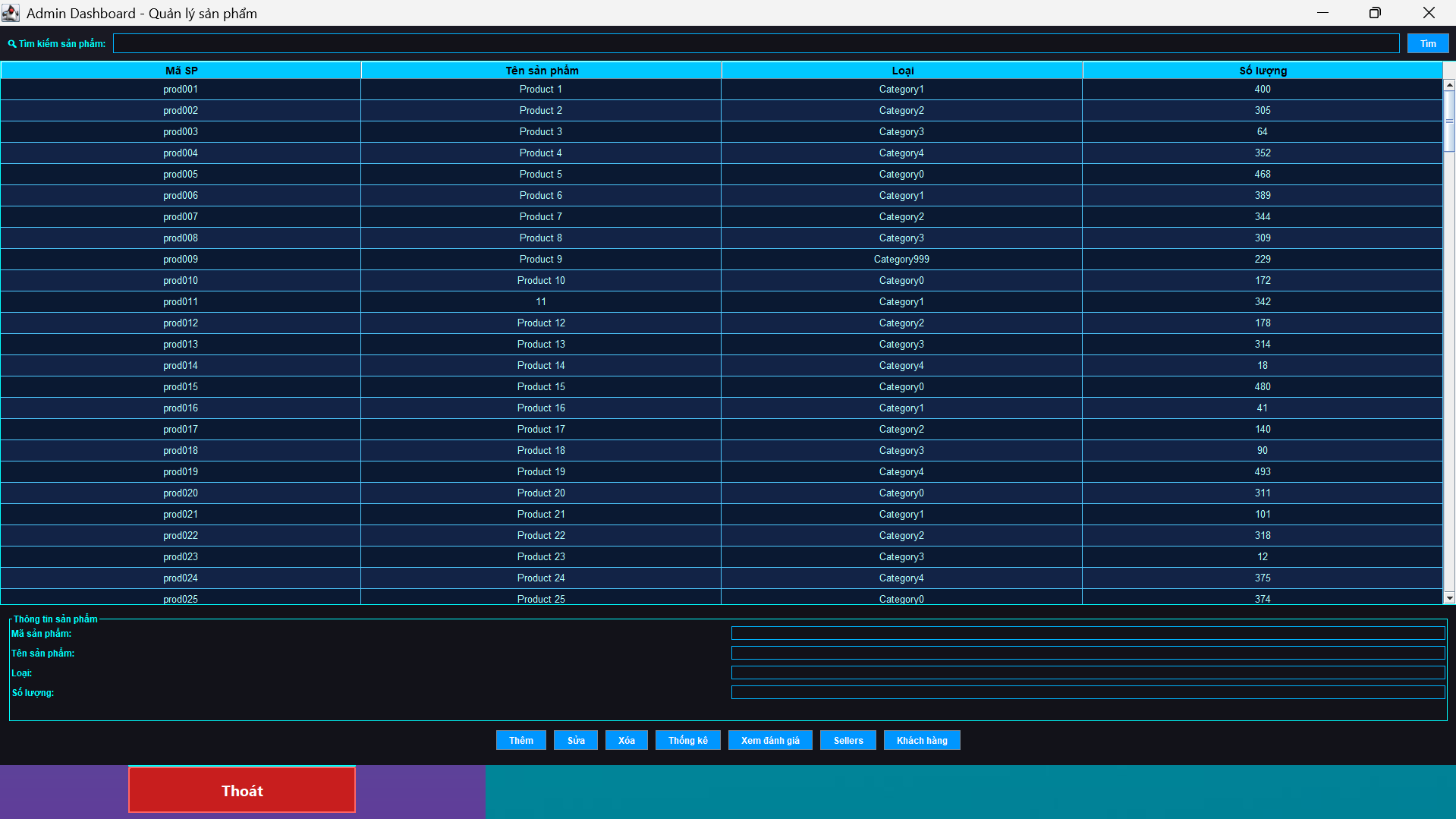
Order management: track status, transaction details.

Revenue statistics with visual charts (JFreeChart).

User management (admin/staff).

Role-based access permissions.

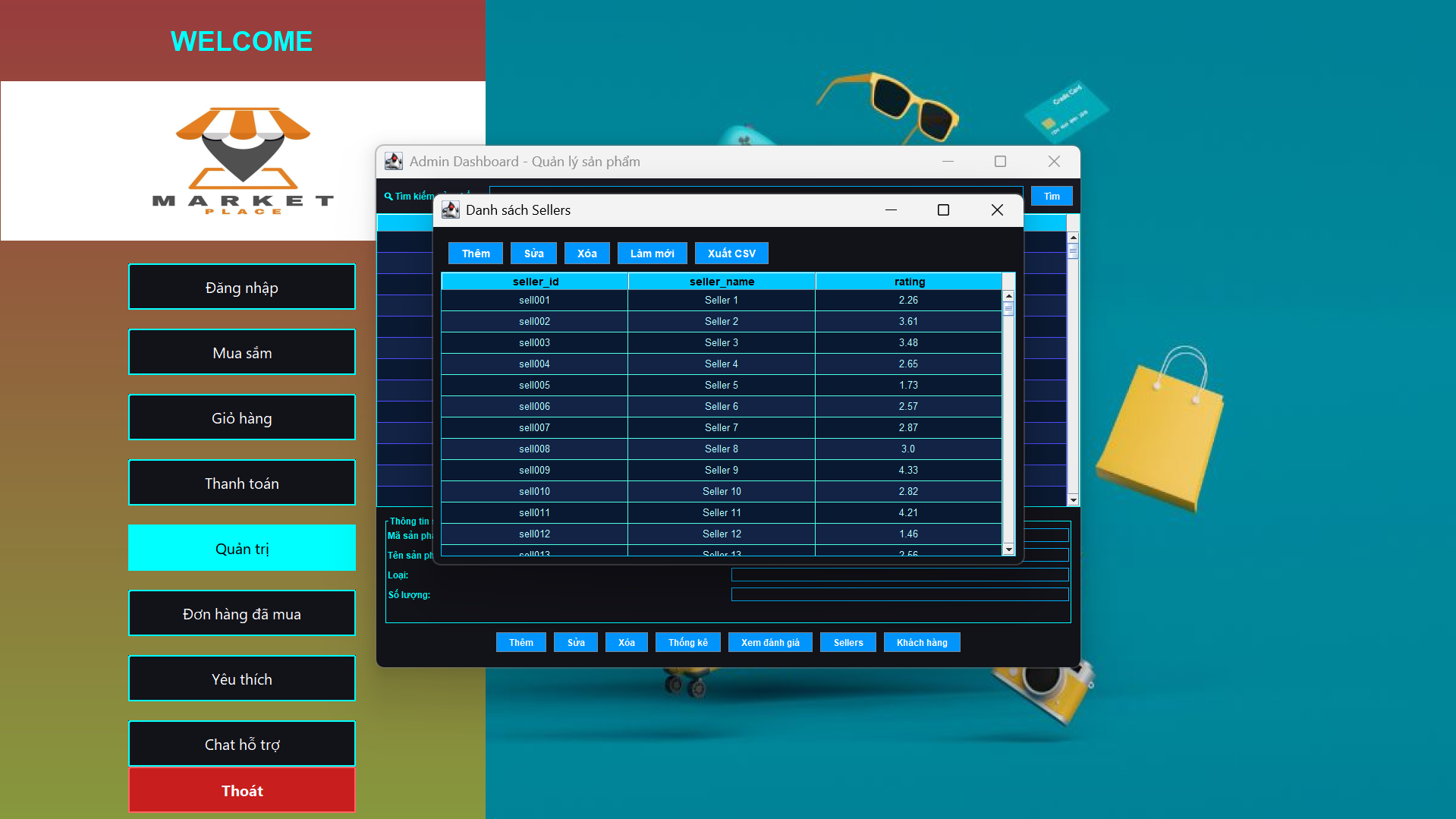
Friendly interface, easy to operate, fast response.



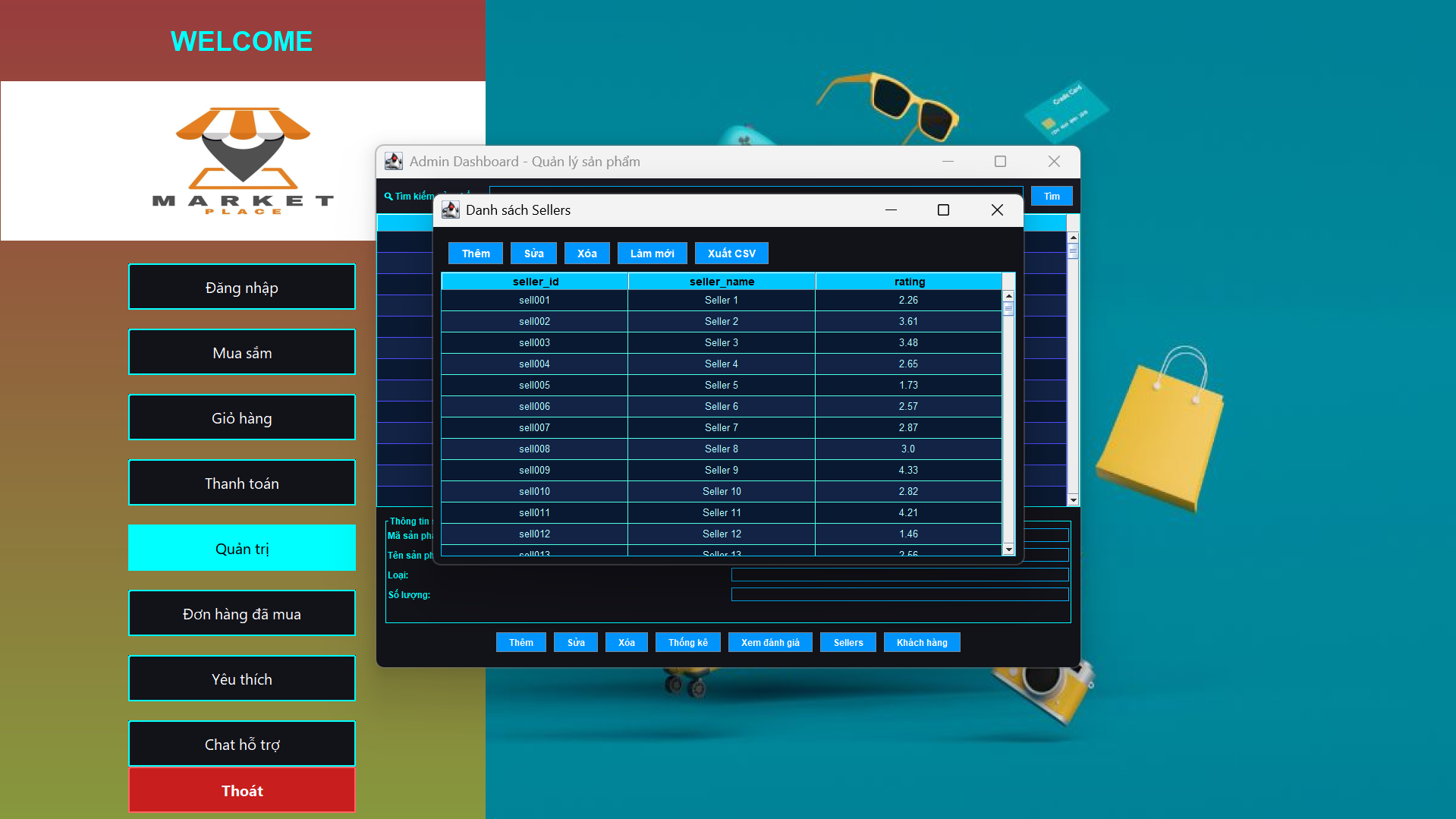
*Figure 1.8 for administrators*



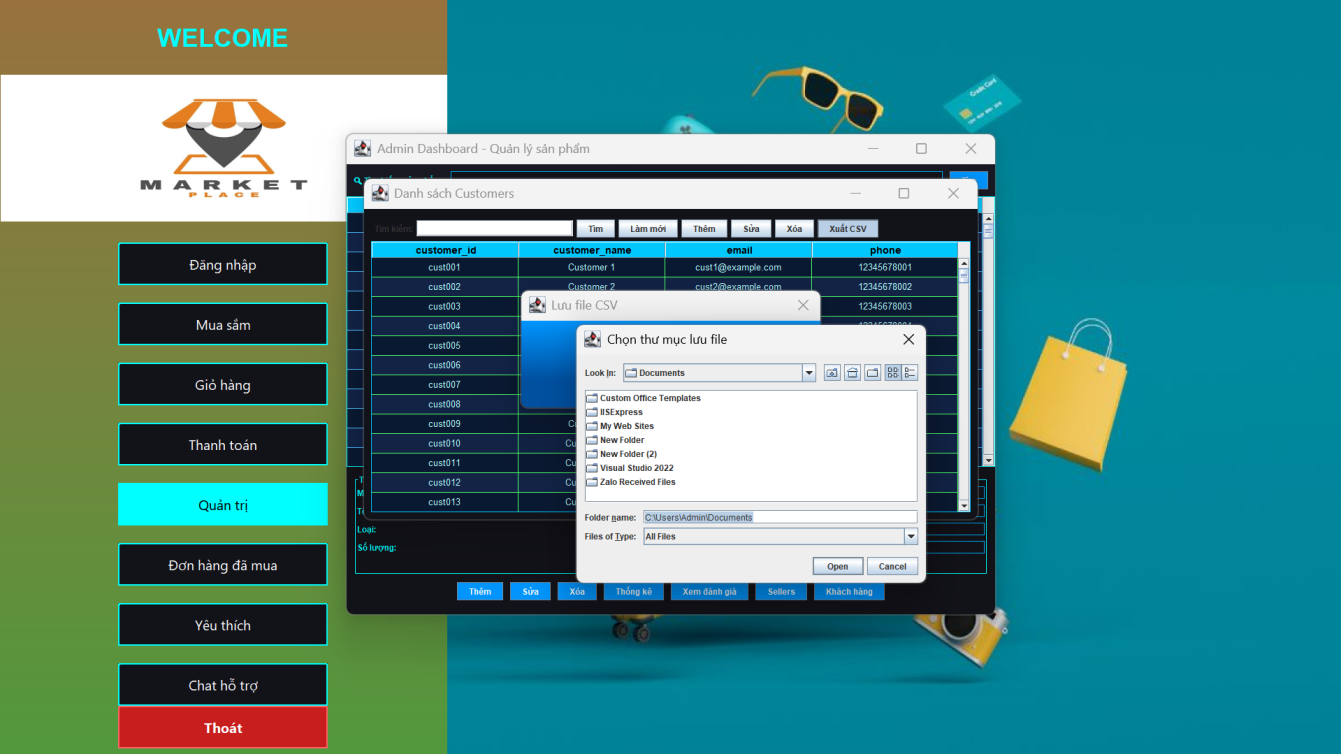
*Dashboard*



*Evaluate*



*Sellers*



*Client*

**2. User interface overview**

AdminDashboardView interface is built with Java Swing, inheriting from JFrame, with layout divided into 3 main parts:

**2.1 Title bar**

Display “Administration System” with large JLabel.

The right corner shows the currently logged in user name and the Log Out button.

**2.2 Left navigation bar**

Use JPanel with vertical layout (BoxLayout), including buttons:

Product Manager (btnProductManager)

Order Management (btnOrderManager)

Revenue Statistics (btnStatistics)

Account Manager (btnUserManager)

Logout (btnLogout)

**2.3 Main display area**

The interface changes content dynamically depending on the selected navigation button.

For example, selecting “Statistics” will display a monthly/yearly sales chart (JFreeChart).

Product management displays JTable of product list and CRUD form.

**3. Source code structure and business processing**

**3.1 Product Management**

Display a list of products in a table (JTable).

Allow Add / Edit / Delete products.

Use ProductDAO to manipulate MySQL (products table).

**3.2 Order Management**

Display order list (JTable).

Allows viewing order details (using OrderItemDAO).

Update order status (e.g. processing → shipped).

**3.3 Revenue statistics**

Use JFreeChart library to draw chart:

Revenue by month/year

Best selling products

Revenue by seller

Data is retrieved from orders, order\_items via SQL JOIN.

**3.4 Authorization and login**

User defined role: admin / staff.

If not admin, hide account management function.

Check permissions when initializing dashboard.

**4. Interaction between layers**

**5. Applied techniques**

OOP:

Apply the layering principle: UI – DAO – Model.

Separate interface and business processing.

Professional Java Swing UI:

Multi-part layout (BorderLayout + CardLayout).

Button icon, background color, hover effect.

JFreeChart:

Create column, pie, line charts from SQL data.

Database Interaction:

JDBC + PreparedStatement.

JOIN query to get revenue and orders.

Session state management:

Save logins and grant access rights.

UI Effects:

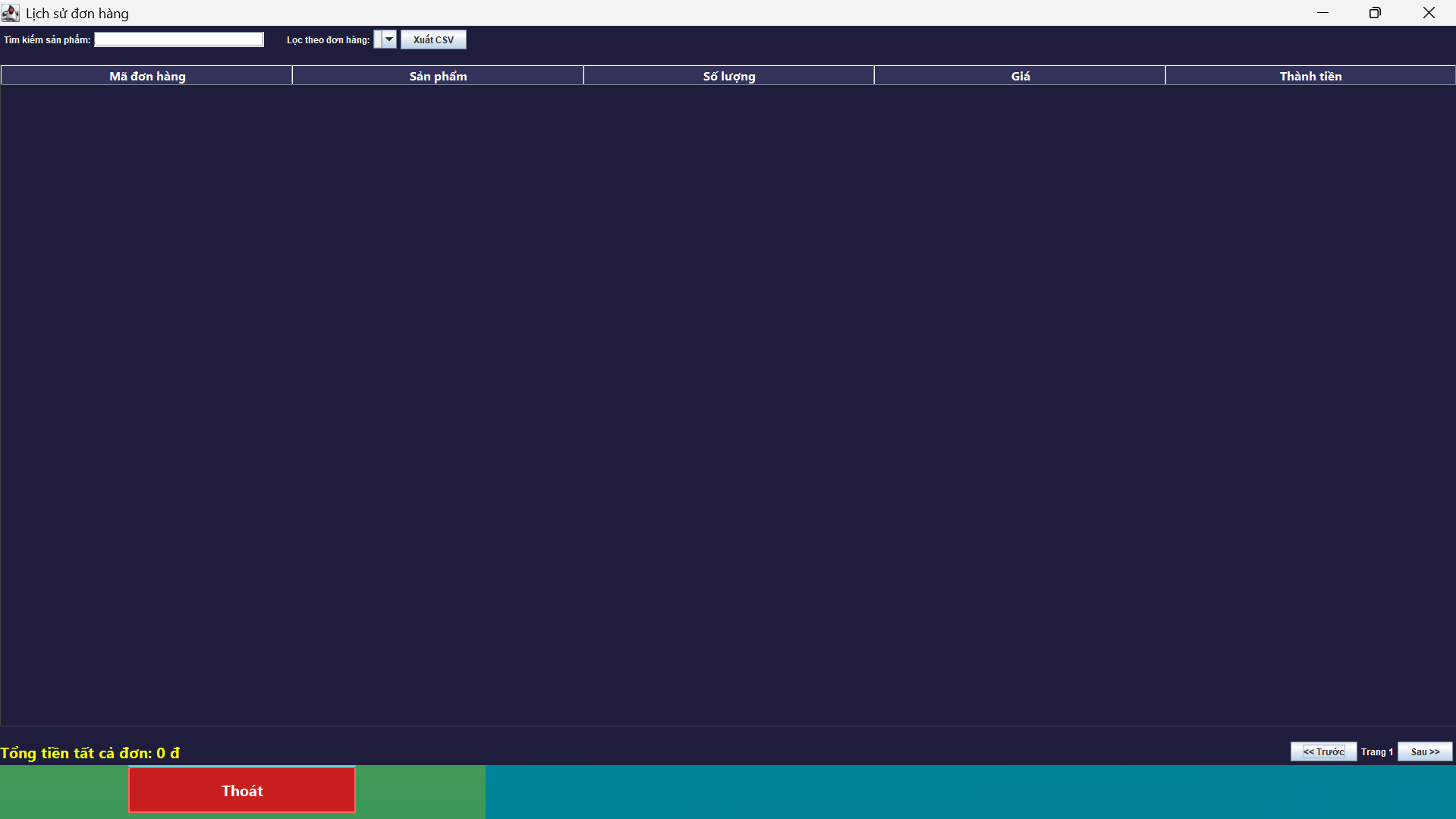
Highlight the selected navigation button.

Confirmation message using JOptionPane.

**2.7.4.7 - Purchased Orders (OrderHistoryView)**

**1. Objective**

The OrderHistoryView interface allows users to review the history of purchased orders, including detailed information about each product in the order, quantity, price of each item, and total payment. Users can search for products, filter orders by order code, paginate to view long lists, and export data to CSV files for storage or sharing.



*Figure 1.9 order history*

**2. Main functions**

Displays a list of products in purchased orders, with columns: Order code, Product, Quantity, Price, Total.

Allows users to search for products by name (type in search box).

Filter the order list by specific order number or choose to view all orders.

Pagination with fixed number of lines per page (15 lines), with navigation buttons between pages.

Displays the total amount of the currently displayed orders.

Button to export current data to CSV file.

Support loading data from database, updating automatic order filter combo.

Error or success messages when loading data, exporting CSV.

**3. Technology used**

Java Swing: building desktop user interfaces.

JDBC (MySQL via DBConnection): query and retrieve data from the database.

DefaultTableModel and JTable to display tabular data.

JFileChooser to choose where to save the CSV file.

Use custom colors and fonts to create a professional, easy-to-read look.

**4. Interface design**

The main background color is dark blue (RGB: 30, 30, 60).

White text, bold table title, with different header border and background color.

The interface distribution includes 3 areas:

Top panel: Search, filter orders, CSV export button.

Center: Scrolling order history table.

Bottom panel: Display total amount and pagination navigation.

The table supports sorting by column, selecting rows with different background colors.

**5. Main operation**

On startup, the interface calls loadOrderItemsFromDB() to query order data from the order\_items table connected to products to get the product name.

The retrieved data is saved in the fullOrderItems list along with the order code list used to create the combo box filter.

Users can enter search keywords or filter by order, the interface will call filterAndDisplay() to filter the data and display the current page.

When changing pages or changing filters/searches, the data displayed on the table will be updated accordingly.

Total amount is calculated based on filtered data.

The CSV export button will save the current table data to a standard CSV file, with column headers.

**6. Pagination and filtering**

Fixed pagination 15 lines per page.

There are "Previous" and "Next" buttons to move between pages.

Display current page number / total number of pages.

Searching by product name is case-insensitive string filtering.

Filter by order based on order code in combo box.

**7. Export data**

Current table data can be exported to CSV file for external storage or analysis.

CSV files have clear headers.

Notification of success or error status when exporting file.

**8. Other notes**

Direct editing on the board is not allowed.

There is exception handling when querying the database or when exporting files.

The interface is designed to be optimized for widescreen (maximized).

There is an addOrderData method that allows adding order data from outside and displays a new order information dialog.

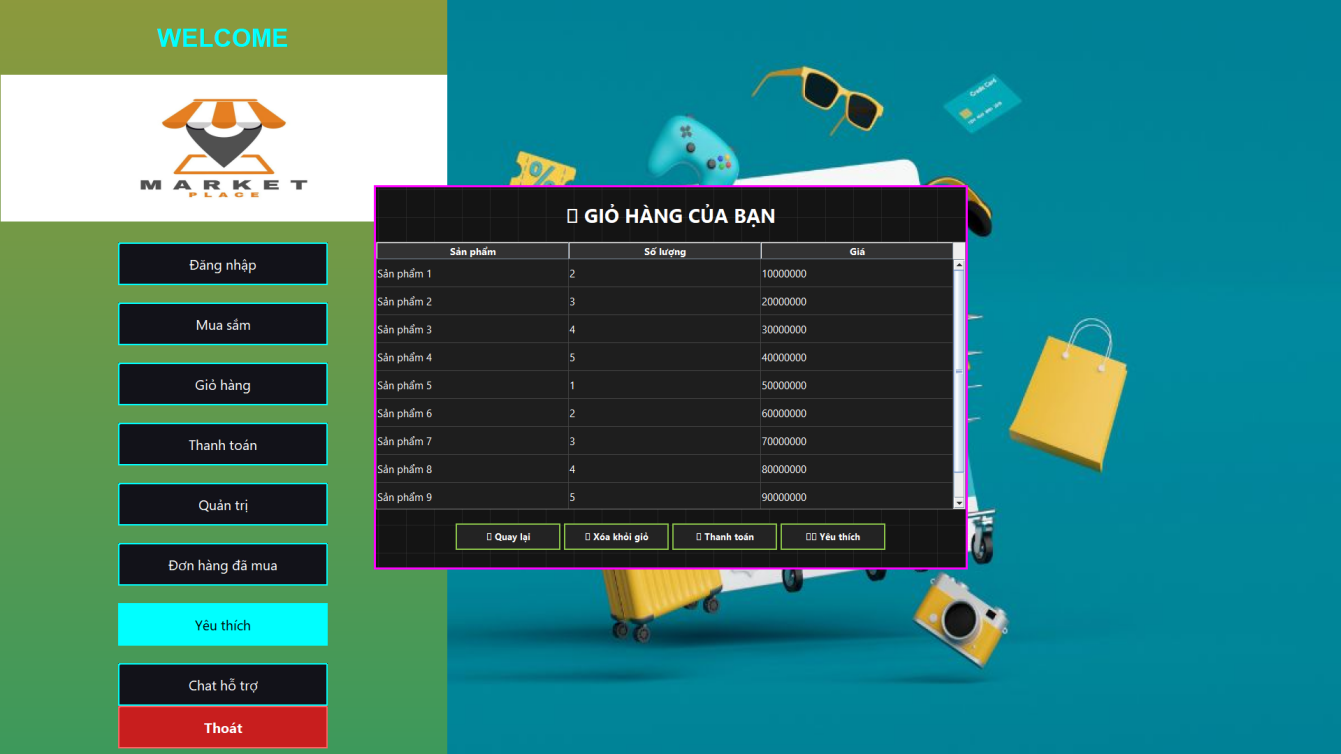
The internal OrderItemDisplay class manages information about each row in the list.

**2.7.4.8 Shopping Cart and Favorites**

**1. Objective**

The Shopping Cart and Favorites functions are an important part of the e-commerce system, helping users manage their selected products effectively and conveniently. The ShoppingCartApp interface is designed to provide a friendly, intuitive and lively user experience, fully supporting common operations such as viewing the list of products in the cart, deleting unnecessary products, making quick payments, and saving favorite products for easy tracking and shopping in the future.

Integrating the Favorites section helps increase user engagement with products, increase the likelihood of customers returning to purchase, and optimize the process of selecting and managing personal products.



*Figure 2.0 list of favorite products*

**2. Main function description**

**2.1. Show cart**

The main interface includes a table listing the products currently in the shopping cart, each line displays:

Product Name: The specific name of the item.

Quantity: Number of products selected.

Price: Price is in the appropriate currency.

The table uses JTable with DefaultTableModel to easily manage, display and update dynamic data.

**2.2. Remove products from cart**

Users can select one or more product lines in the table and then press the "🗑Remove from cart" to remove unwanted products.

The system checks to see if any rows are selected, otherwise a warning dialog box is displayed asking to select a product before deleting.

**2.3. Payment**

When pressing the " button💳Checkout", the system will calculate the total amount by multiplying the quantity by the price of each product in the cart.

The payment interface (CheckoutView) is initialized with the total amount transmitted, the user will perform the next steps such as entering payment information, shipping address,...

After opening the checkout interface, the current shopping cart interface is hidden to avoid duplication or mistaken operations.

**2.4. Add product to wishlist**

The user can select one or more products in the shopping cart table and press the "❤️ Favorites" to add those products to your favorites management list.

The products are sent to the FavoriteManager class for storage or further processing.

The system will display a confirmation message that the products have been successfully added to the favorites list.

**3. Interface design and user experience**

**3.1. Overall interface**

The main window is 850x550 pixels in size, without the traditional title bar (undecorated), making it easy to customize effects and colors.

The window appears in the center of the screen for a friendly and easy-to-see feel.

**3.2. Color and style**

Apply Dark Mode theme with dark background (dark gray), white text, creating good contrast, easy to read and reducing eye strain when used for a long time.

The window border uses a gradient color effect that sequentially transitions through Cyan, Magenta, Orange, and Pink to create a lively, modern look.

The buttons have a continuous border color change effect, creating highlights, helping users to easily distinguish and interact.

**3.3. Product table**

The data in the table is clearly presented, with Segoe UI font size 15pt, line height 40 pixels, easy to follow.

The board background color is dark gray, the selected line has a prominent dark blue background color, and the text is white, providing an intuitive experience.

Dark gray table grid lines increase separation of data cells.

Table titles use bold font, dark background, white text to help quickly identify data columns.

**3.4. Function buttons**

Four main buttons: "⬅Come back", "🗑Remove from cart", "💳Pay", "❤️ Favorite" are all appropriately sized, 13pt bold font, white text stands out on a dark background.

These buttons have a smooth, dynamic border color-changing effect, creating a lively feeling, helping users easily operate and recognize functions.

**3.5. Dynamic Background**

Use AnimatedGridPanel to draw a grid background that moves smoothly in cycles, creating a modern and unique effect for the interface.

The grid color is matte gray, in harmony with the overall Dark Mode, not causing glare.

**4. Architecture and implementation techniques**

**4.1. Using Java Swing**

The entire interface is built using the Swing library, providing detailed customization and control of each UI element.

Use JFrame as main window frame, JTable to display data list, JButton for function buttons.

**4.2. Table data management**

Shopping cart data is managed using a DefaultTableModel with fixed columns.

Table mode is set to disallow direct editing (protect data integrity).

Sample data is loaded using the loadSampleData() method during initialization for testing and demoing functionality.

**4.3. Dynamic effects and events**

Use javax.swing.Timer to create a cyclic color change effect for interface borders and buttons.

Another Timer is used to draw and update the grid background dynamically, adding liveliness to the interface.

ActionListener-attached buttons handle user interactions: clearing a selection, moving to checkout, adding to favorites, or exiting the interface.

**4.4. Functional separation**

The shopping cart interface and favorite product management are clearly separated.

FavoriteManager is a private class responsible for storing and managing the list of favorite products, making it easy to extend and maintain.

**5. Usage process**

The user opens the shopping cart interface, the product list is fully displayed.

Users can select one or more products in the table to operate.

If you want to remove a product, select the product and press the "Remove from cart" button.

When you want to pay, click "Pay" to see the total amount and proceed to the next steps.

If you want to save a product for easy access in the future, select the product and click "Favorites".

To return to the previous screen, press "Back".

**6. Proposed improvements and expansions**

Persistent Favorite Storage: Currently FavoriteManager stores temporary, database-integrated favorites lists for each user account.

Live cart update: Allows editing of quantities in the cart table instead of just viewing and deleting.

Notification and confirmation: Add confirmation dialog when deleting product or checkout to avoid wrong operation.

Optimized performance: Reduce resources used by Timers by centrally managing and updating the interface more efficiently.

Network Integration: Sync shopping cart and favorites with server for multi-device support and security.

Responsive interface: Optimize the interface to support different screen sizes and devices.

**3.4.8 Shopping Cart and Favorites**

**1. Objective**

The Shopping Cart and Favorites functions are essential components in the e-commerce system, helping users manage selected products conveniently and effectively. The ShoppingCartApp interface is designed to provide a friendly, intuitive, and lively experience, fully supporting commonly used operations such as:

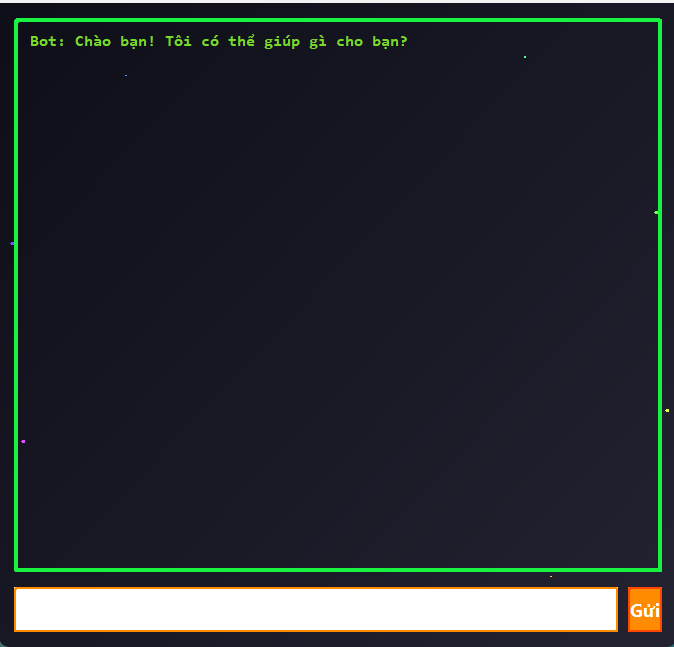
View list of products in cart,

Delete products that are no longer needed,

Make payments quickly,

Save your favorite products for easy tracking and shopping later.

The addition of the Favorites section aims to increase user engagement with products, encourage customers to return to purchase, and optimize the process of selecting and managing personal products.



*Figure 2.1 support chat*

**2. Main function description**

**2.1. Show cart**

The main interface includes a table listing all products in the shopping cart.

Each line displays the following information:

Product Name: specific item name,

Quantity: number of products selected,

Price: unit price in appropriate currency.

The table uses JTable combined with DefaultTableModel for easy management, display and dynamic updating of data.

**2.2. Remove products from cart**

Users can select one or more product lines from the table.

Press the " button🗑Remove from cart" to remove unnecessary products.

The system checks if no line is selected and displays a warning dialog asking to select a product before deleting.

**2.3. Payment**

When pressing the " button💳Checkout", the system calculates the total amount by multiplying the quantity by the price of each product in the cart.

The CheckoutView is initialized with the total amount passed in.

Users perform the next steps such as entering payment information, shipping address,...

After opening the checkout interface, the current shopping cart interface will be hidden to avoid duplication or mistaken operations.

**2.4. Add product to wishlist**

The user selects one or more products in the shopping cart and presses the "❤️ Like".

The products are sent to the FavoriteManager class for storage or further processing.

The system displays a confirmation message that the product has been successfully added to the favorites list.

**3. Interface design and user experience**

**3.1. Overall interface**

The main window is 850x550 pixels in size, without the traditional title bar (undecorated), allowing flexible customization of effects and colors.

The window appears in the center of the screen, creating a friendly and easy-to-see feel.

**3.2. Color and style**

Apply Dark Mode theme with dark background (dark gray), white text creates high contrast, easy to read and reduces eye strain when used for a long time.

The window border uses a gradient color effect that sequentially transitions through Cyan, Magenta, Orange, and Pink, creating a modern and vibrant feel.

The buttons have a continuous border color change effect, creating highlights, helping users to easily distinguish and operate.

**3.3. Product table**

The data in the table is clearly presented with Segoe UI font, size 15pt, line height 40 pixels, easy to follow.

The background color of the table is dark gray, the selected row has a prominent dark blue background, and white text.

Dark gray table grid lines help separate data cells clearly.

Table titles use bold font, dark background, white text to help quickly identify data columns.

**3.4. Function buttons**

There are 4 main buttons: "⬅Come back", "🗑Remove from cart", "💳Pay", "❤️ Like".

Appropriate button size, 13pt bold font, white text stands out on dark background.

The continuous border color changing effect makes the button vivid, easy to recognize and operate.

**3.5. Dynamic Background**

Use the AnimatedGridPanel to draw a grid background that moves smoothly and periodically.

The matte gray mesh color is in harmony with the overall Dark Mode, not causing glare.

**4. Architecture and implementation techniques**

**4.1. Using Java Swing**

The entire interface is built using the Swing library, providing detailed UI customization.

Use JFrame as main window frame, JTable to display list, JButton for function buttons.

**4.2. Table data management**

Shopping cart data is managed using a DefaultTableModel with fixed columns.

Table mode does not allow direct editing to protect data integrity.

Sample data loaded using loadSampleData() method during initialization for demo.

**4.3. Dynamic effects and events**

Use javax.swing.Timer to create a periodic color change effect for interface borders and buttons.

Another timer is used to update the dynamic grid background, adding liveliness to the interface.

Buttons attached to ActionListener handle interactions: delete row, go to checkout, add favorite, or exit.

**4.4. Functional separation**

The shopping cart and favorites management interfaces are clearly separated.

The FavoriteManager class is responsible for storing and managing the favorites list, supporting easy extensibility and maintenance.

**5. Usage process**

The user opens the shopping cart interface, the product list is fully displayed.

You can select one or more products to operate.

To remove a product, select it and press "Remove from cart".

When checking out, click "Checkout" to view the total and enter the required information.

To save a product for future purchase, select and click "Favorite".

To return to the previous screen, press "Back".

**6. Proposed improvements and expansions**

Persistent Favorites Storage: Integrate database to save favorites list by user account.

Update cart directly: Allows editing of quantities directly in the cart table.

Notification and confirmation: Add confirmation dialog when deleting products or making payments to avoid mistakes.

Optimized performance: Centralized management of Timers, more efficient interface updates.

Multi-device sync: Connect to server to sync shopping cart and favorites, support multiple devices.

Responsive interface: Optimized display on different screen sizes and devices.

**2.3 System operation diagram and interface — Marketplace Transactional**

**1. System Activity Diagram**

**1.1. Overview of system operations**

The Transactional Marketplace system is an e-commerce platform that connects buyers and sellers in an online trading space. Users can browse products, manage shopping carts, make payments and track orders, while the system ensures accurate data processing, security and smooth operations.

The system has the following main types of users:

Customer: The person who buys the product.

Seller: The person who posts the product for sale.

Administrator (Admin): Manage the entire system.

**1.2. Main operating procedure of the system**

The process is divided into the following steps:

**Step 1:**Login / Register

User access to the system.

Select login if you already have an account, or register a new one.

The system authenticates user information and assigns corresponding permissions (Customer, Seller, Admin).

**Step 2**: Browse and search for products

User visits product listing page.

Supports filtering by category, price, rating, brand.

Users can search quickly using the search bar.

**Step 3:**Cart Management

Add products to cart from the list or product detail page.

Display quantity and price of each product in the cart.

Allows editing quantity or removing products from cart.

**Step 4:**Payment and Order

User confirms products in cart.

Enter your shipping address and payment method.

The system performs inventory checks and calculates total money.

Send payment request via Payment Gateway.

Receive payment results and order status updates.

**Step 5:**Track orders and receive notifications

Users can track order status through the following statuses: Processing, Shipping, Delivered, Canceled.

The system sends automatic notifications via email or app.

**Step 6:**Product and order management (Admin & Seller)

Sellers and admins manage product lists: add new, edit, stop selling.

Manage orders, process return and refund requests.

Revenue statistics, business performance.

**1.3. Detailed operation diagram**

| **Work** | **Interpretation** | **The performer** |
| --- | --- | --- |
| System access | Open the Marketplace Transactional app or website | User |
| Login or Register | Enter account information, password or create new account | User |
| Access authentication | Login and authorization verification system | System |
| Access product list | Display product categories, support filtering and searching | User |
| View product details | Show description, images, reviews, prices | User |
| Add product to cart | Select quantity, add product to cart | User |
| Cart Management | Show products, edit quantity, delete products | User |
| Make payment | Enter payment information, confirm order, connect payment gateway | User |
| Update order status | The system updates order status according to each processing step. | System |
| Order Tracking | View order status and transaction history | User |
| Product and order management | Add/edit/delete products, process orders, reports | Seller/Admin |

**1.4. Special situations**

**Payment failed:**The user is notified of an error, asked to try again or choose another method.

**Out of stock:**When a user places an order, the system checks inventory, if out of stock, it immediately notifies and does not allow payment.

**Cancel order:**Users or admins can cancel orders under certain conditions.

**Multi-device management:**Users can log in on multiple devices, sync shopping carts and history.

**Activity diagram illustration (flow description):**

User

├─> Login / Register

├─> Browse products

├─> Add product to cart

├─> View cart

├─> Payment

├─> Receive order confirmation

├─> Personal order management

System

├─> User authentication

├─> Product and inventory management

├─> Cart Processing

├─> Order processing

├─> Submit payment information

Admin

├─> Product Management

├─> Order Management

├─> User Management

**2. System interface diagram (UI Flow Diagram)**

**2.1. Interface overview**

The Marketplace Transactional system is designed with a friendly, easy-to-use interface, supporting multiple platforms (web and mobile applications). The interface is divided into main modules:

Login and registration page:

Provides simple, highly secure data entry forms with password recovery support.

Product list page:

Display products in grid or list, with search bar and detailed filters.

Product detail page:

Display images, details, reviews, comments, add to cart button.

Cart:

The table displays the selected product, quantity, price, total, delete button and checkout button.

Pay:

Form to enter shipping information, payment method, review order and confirm.

Order history:

List of placed orders, detailed status, option to cancel or contact support.

Admin & Seller:

Dashboard overview, product management, orders, reports, user management.

**2.2. Detailed interface navigation flow**

| **Current screen** | **User Actions** | **Next screen** |
| --- | --- | --- |
| Login Screen | Login successful | Product list page |
| Login Screen | Select Sign Up | Registration Page |
| Registration Page | Complete registration | Login Page |
| Product list page | Select product to see details | Product detail page |
| Product list page | Add product to cart | Keep page intact |
| Product detail page | Add product to cart | Keep page intact |
| Product list page | Select cart | Cart page |
| Cart page | Edit quantity / Delete product | Update cart |
| Cart page | Click Pay | Checkout Page |
| Checkout Page | Confirm and make payment | Order confirmation page |
| Order confirmation page | View order status | Order History Page |
| Order History Page | Select order to view details | Order detail page |
| Admin Dashboard Page | Product / order / user management | Corresponding page (Product and order management) |

**2.3. Interface of each screen**

Login screen:

Input fields: Email/Username, password.

Buttons: Login, New Registration, Forgot Password.

Login error message.

Product list page:

Grid product list with photo, name, price, Add to cart button.

Left filter: by type, price, brand, review.

Top search bar.

Product detail page:

Large images, detailed descriptions, technical information.

Add to cart button, Favorites button.

Customer reviews and comments section.

Cart:

Product table: name, quantity (editable), price, total.

Delete individual product button.

Payment Button.

Pay:

Delivery address entry form: name, address, phone number.

Choose payment method: bank transfer, credit card, e-wallet.

Review total amount, shipping charges.

Confirm Order button.

Order history:

List of orders with status (processing, shipping, completed).

View details button, cancel order if allowed.

Status update notification.

Dashboard Admin / Seller:

Overview dashboard.

Left menu with management functions.

Charts of sales, orders, best-selling products.

Interface diagram illustration (screen switching)

[Login/Register]

↓

[Product Home] ←→ [Product Details]

↓ ↓

[Cart] ←───────────────┘

↓

[Pay]

↓

[Order Confirmation & History]

(Admin Dashboard)

├─> Product Management

├─> Order Management

└─> User Management

**CHAPTER 3: SYSTEM OPERATING MECHANISM**

**3.1. Overall architecture of the system**

The transactional marketplace system is developed in Java language, using Java Swing interface library and database connection mechanism via JDBC. The project is organized according to MVC (Model - View - Controller) model, which helps to clearly separate data, interface and logic processing.

Model: Includes classes that represent data such as User, Product, Order, store attributes and related processing logic.

View: Swing classes like Home, CheckoutView, CartView, CustomerDashboard, are responsible for displaying information and user interface.

Controller: Includes controller classes such as CartManager, ShoppingCartApp, MainApp, which handle user actions and navigate between screens.

The system structure is clearly divided into packages and named according to features, helping to increase scalability and maintainability.

**3.2. User authentication and authorization mechanism**

**3.2.1. Login and register an account**

One of the core mechanisms that ensures security and decentralization in a transactional marketplace system is the user authentication process, which includes two functions: login and registration. These two functions are implemented through the interface classes LoginForm.java and RegisterForm.java, and interact directly with the database through the class DBConnection.java.

***a) Login interface and logic – LoginForm.java***

The LoginForm interface provides the user with two input fields: username and password. After filling them in and pressing the "Login" button, the following process is triggered:

Get input data:

The system takes input strings from text fields (JTextField for username and JPasswordField for password).

Performs whitespace removal and empty string checking to ensure valid input.

Connect and check account:

LoginForm uses DBConnection.getConnection() to open a connection to the database.

Execute SQL query:

select \* from users where username = ? and password = ?

Validate query results:

If the result returns one row of data, then the account exists and is valid:

The data is mapped to a User object containing the following properties:

userId: User identifier

username: Login name

password: Original password

role: Permissions (customer/admin)

If no result: display login error message.

Redirection by permission:

If role = customer, the system opens the CustomerDashboard interface.

If role = admin, the system opens AdminDashboard.

Save login state:

The User object after successful login is passed to other interfaces (via constructor or static variable) to serve as the basis for operations in the session.

***b) Registration interface and logic – RegisterForm.java***

RegisterForm is an interface that allows users to create new accounts. Users enter information including:

Username

Password

Confirm Password

Full name, phone number, email, etc. (optionally extended)

The processing procedure is as follows:

Check input data:

Compare password and confirm password to make sure they match.

Check if the login name already exists using the command:

select \* from users where username = ?

Create new account:

If the username does not exist:

Send an INSERT INTO query to add users to the users table.

The default permission is "customer".

Create new account:

If the username does not exist:

Send an INSERT INTO query to add users to the users table.

The default permission is "customer".

Announcement of results:

If registration is successful: display dialog “Account created successfully”.

If failed: appropriate error message (e.g. account already exists).

Navigation after registration:

The system can automatically redirect users to the LoginForm interface to perform the first login.

***c) User class and authorization***

The User class is the central data model in the system. It defines the data structure for every user, including:

public class User {

private int userId;

private String username;

private String password;

private String role;

// getter, setter, constructor, etc.

}

After a user successfully logs in, this User object is used to:

Define access rights to functions (purchase, administration, view statistics).

Attach the orderer information to the order (Order.userId)

Display username in interfaces (Homepage, CustomerDashboard, AdminDashboard)

***d) Improved security – password encryption***

Currently the system is saving passwords in plaintext form – which is not secure. To improve security, one of the following methods can be applied:

Using MD5 or SHA-256 hash function:

Before saving the password to the database, convert the string to a hash:

String hashedPassword = DigestUtils.sha256Hex(plainPassword);

When logging in, the user enters the password → also hashed → compared with the saved hash.

Add “salt”:

Each user has their own random string (salt) → increasing the difficulty of cracking with a dictionary.

Store passwords using BCrypt or PBKDF2:

Integrate libraries like BCrypt (Spring Security) for encryption.

***e) Scalable advanced features***

| **Features** | **Describe** |
| --- | --- |
| **Password Recovery** | Send OTP code or link via email to reset password |
| **Two-Factor Authentication (2FA)** | Use OTP code via email/phone number when logging in |
| **Login Logging** | Save time, IP, access device to enhance security |
| **Block attack account** | If you log in incorrectly too many times in a row, the system will temporarily lock your account. |

**3.2.2. User authorization**

After a user successfully logs in, the system relies on the role attributes of each account to route and determine access rights. Specifically:

If the role is "customer", the user will be redirected to the main customer interface (CustomerDashboard), where they can view products, add to cart, place orders, track orders, and rate products.

If the role is "admin", the user is redirected to the admin interface (AdminDashboard), where they have access to all system functions such as managing products, sellers, users, statistics, processing reviews and controlling orders.

Authorization is applied not only at the time of interface navigation but also checked throughout the user's actions. Actions such as adding products, accessing statistical dashboards, or modifying system information will be denied if the user does not have sufficient authority, ensuring security and clear separation between roles.

**3.3. Display products and manage shopping cart**

**3.3.1. Home page and product list**

The main interface of the system is where the list of products for sale is displayed. All data is loaded from the database via dynamic connection. After the data is retrieved, the system will build a product table including visual components such as: representative image, product name, price, inventory quantity, and function buttons such as "Add to cart", "View details".

Users can use keyword search functions, filter products by specific categories, and use smooth transitions to navigate between product pages. Animations, color transitions, and a modern look are all handled by interface tools like Java Swing combined with custom graphics.

When a user selects a product and clicks “Add to Cart”, the product information is sent to an intermediate shopping cart management layer, which temporarily stores information about the selected items.

**3.3.2. Shopping cart and updates**

The shopping cart information is managed by a middleware layer, which stores all the products that have been added from the home page. The shopping cart interface will display a list of these products in a table, where the user can perform the following operations:

Update desired product quantity

Remove product from cart

View total selected cost

Any operation performed at the interface will also synchronize the data in the intermediate memory to ensure consistency. The system will automatically update the total payment amount in real time whenever the product quantity changes.

After completing the selection, the user can go to the payment interface to make the order.

**3.4. Payment and order processing mechanism**

**3.4.1. Payment interface**

The payment interface provides full information fields for users to enter necessary data such as: recipient's full name, delivery address, phone number. Then, users can choose payment methods including:

Cash on delivery

Pay by QR code

When the user chooses the QR form, the system will generate a QR code corresponding to the total order amount and display it on the interface, helping to link with e-wallets or banking applications in the upgraded version.

The system will automatically calculate related costs such as: tax, shipping fee, or discount if the user enters a promotional code. When the user clicks “Confirm Order”, all order information will be recorded in the database, creating a new order record with the initial status of “Waiting for confirmation”.

**3.4.2. Order history**

After successfully placing an order, users can review all orders made through the order history interface. The interface allows filtering by:

Order time

Order status: pending, shipping, completed

Each order will have detailed information such as product list, total amount, shipping cost, discount code (if any), order status and creation date. Users can easily track the shipping progress, feedback or file a complaint in case of an order problem.

**3.5. Enhanced Features and Experience**

**3.5.1. Real-time support chat**

The system integrates a simple chat interface that allows users to communicate with technical support or a virtual assistant. The current version simulates a basic conversation, but is designed to be open for easy upgrades with features such as:

Multi-user group chat

Integrate real-time chat using WebSocket or RabbitMQ technology

Save chat history for reference and evaluation

**3.5.2. Discount codes and QR payments**

Users can enter a discount code during checkout. The system will check the code's eligibility criteria such as expiration date, usage limit, and eligible users. If valid, the discount will be deducted directly from the total payment amount.

With QR code payment, a QR code is generated corresponding to the amount to be paid and can be integrated with services such as Momo, ZaloPay, VNPAY in the future. This facilitates a cashless, safe and modern payment experience.

**3.6. System statistics and administration dashboard**

**3.6.1. Dashboard Interface**

The system provides a management dashboard for administrators with an intuitive interface displaying:

Revenue by month or year

Pie chart showing the percentage of product categories sold

Line chart tracking user and order growth

Total number of products available, number of sellers, total revenue over time

All charts are built using a dedicated graph library, supporting vivid, easy-to-understand and interactive displays.

**3.6.2. Filters and interactions**

Dashboard allows filtering data by:

Specific period of time (month, quarter, year)

Specific Seller

Product Catalog

Filtering will directly affect the data in the chart, helping administrators easily grasp the business situation from different perspectives. In addition, it is possible to integrate the function of exporting PDF or Excel reports for storage and in-depth analysis.

**3.7. Database and transaction management**

**3.7.1. Database connection**

All data access activities in the system are done through an intermediate connection layer that is responsible for:

Create a secure connection to the database

Query the users, products, orders, and cart tables

Ensure that connection resources are always closed after each use to avoid resource leakage

**3.7.2. Transaction**

During the ordering process, the system initiates a complete transaction. That is, all related data such as order, product, discount information, etc. will be recorded simultaneously. If any operation during the recording process fails, the entire transaction will be canceled (rollback), ensuring that there is no data corruption or loss.

**3.8. System scalability**

With its modular architecture and clear interface, the system has great potential for future development. Some expansion directions include:

Integrate electronic payment gateways such as Momo, ZaloPay, VNPAY

AI-powered product recommendations, based on purchase history or search behavior

Sentiment analysis in product reviews to detect potential problems

Automatically send personalized coupon codes

Build REST APIs to connect to mobile apps or partner platforms

Expanded shipping management features, supporting real-time delivery status tracking and automatic notifications to users.

**Chapter 4: Product testing and evaluation**

**4.1. Experimental objectives**

The objective of the experiment is to test the completeness, efficiency and real-world performance of the developed transactional marketplace system. The experiment is conducted on a set of key functions such as login/registration, user authorization, product display, shopping cart, payment, order management and system administration functions. In addition, the evaluation also considers factors related to performance, stability, user interface and real-world user experience.

**4.2. Experimental environment**

The system is deployed and tested on a personal computer platform using Windows 10 operating system. The software is developed in Java using the Eclipse IDE environment. The database management system is MySQL, with the stored data being manually initialized and gradually expanded according to the usage scenario. Supporting libraries such as JFreeChart, JDBC and Swing library are fully integrated to ensure smooth operation.

The system is deployed and tested in a simulated environment with the following configuration:

Hardware: CPU: Intel Core i5-1135G7 RAM: 8GB

Hard drive: 256GB SSD

Software: Operating System: Windows 11 64-bit IDE: Eclipse IDE for Java Developers

Database: MySQL 8.0

Charting tool: JFreeChart

Java Runtime: Java SE 17

Sample data: 100+ products in 5 different categories 20 user accounts (including admin and customers) 50 sample orders

The test subjects included the development team and a number of simulated users, including novices who had never used similar software before. The evaluation focused on two main groups: regular users (customers) and system administrators.

**4.3. Experimental procedure**

The experimental process is conducted through specific steps:

**4.3.1. Testing registration and login functions**

The test user creates a new account with different information to test the validity of the system. Some situations such as missing information, incorrect password, or using an existing account are tested to verify the input data checking feature.

After successful registration, the user logs in to check the authentication process, checking for incorrect username or password to ensure the system responds correctly to the error message. The system has fully handled the situations and directed the user to the correct interface according to the role.

**4.3.2. Testing authorization and interface navigation**

After logging in, users are directed to the appropriate interface: administrators are directed to the management dashboard; regular users are directed to the customer interface. Access to specific functions for each role (such as adding products, viewing statistics, editing sellers) is clearly controlled.

Testing for unauthorized access to unauthorized features (e.g., regular users accessing the admin page) was also conducted and was properly prevented by the system.

**4.3.3. Display and search products**

Users tested the ability to load products from the database, including images, prices, descriptions, and action buttons. The search functionality by product name and filtering by category was tested with various keywords.

Interface effects such as transitions, hover effects and light/dark mode effects are also positively evaluated by users, helping to enhance the user experience.

**4.3.4. Shopping cart and updates**

When the user clicks “Add to Cart”, the product is successfully added to CartManager and displayed in CartView interface. Functions such as editing quantity, deleting products, calculating total order value are operated smoothly.

The system ensures synchronization between the shopping cart data on the interface and the intermediate processing layer. Any changes are reflected immediately, creating a smooth feeling for the user.

**4.3.5. Make payment**

Users test the full checkout process with filling in shipping information, choosing a payment method, applying a discount code, and confirming the order.

The system automatically calculates the total amount, tax, discount, and saves the order information to the database. The QR Code generation function works well, the code is displayed clearly and reflects the correct value to be paid.

Tests with missing and misformatted information are also performed to ensure the system responds appropriately and does not leak erroneous data.

**4.3.6. Order history and tracking**

Users can view all orders placed, filtered by time and status. Details of each order include product list, shipping cost, order time, and delivery status.

Checking orders with different statuses like “Waiting for confirmation”, “Delivering” and “Completed” showed that the system responded accurately and the interface was easy to operate.

**4.3.7. Statistics and administrative dashboard**

Admins access the dashboard to view charts of revenue, best-selling products, number of users, products, and orders. Filtering by time and seller is tested with different data.

Charts are updated automatically, operate smoothly and are error-free. Pie charts and line charts provide a vivid and intuitive view.

**4.3.8. Enhanced support and experience features**

Simulated chat, coupon entry, and QR code payment were tested in multiple scenarios. Users appreciated these additional features as they made the system feel complete and modern.

The feature of suggesting products based on preferences is not yet integrated, but the system is designed to be ready for future development.

**4.4. Performance evaluation**

The system operates stably with small to medium number of users. With the test database containing about 100 products, 30 users and more than 50 orders, queries and operations all take place in reasonable time, without any hangs or delays.

System performance can be further improved by optimizing queries, using connection pools, and adding caching for common functions.

**4.5. User feedback**

Simulation users have provided a lot of positive feedback, especially regarding the ease of use of the interface, the clarity of the functions, and the responsiveness of the system. Some suggestions for improvement include:

Add user guide for beginners

Enhance password security with encryption

Allow saving cart status when exiting the program

Improved interface layout for small screen devices

**4.6. Summary of experimental results**

The transactional marketplace system has achieved most of the goals set out during the development process. The main functions all work well, the interface is friendly, and the extensibility is high. The source code is organized in a clear model and divided into reasonable layers, making maintenance and upgrading in the future easy.

Although there are some minor limitations and not all advanced features are yet to be integrated, the current system is qualified for real-world testing and further refinement into a finished product.

**CONCLUSION**

**Results achieved:**

The research and development project of the "Marketplace Transactional System" application has completed all the set goals with high quality and efficiency, successfully building a multi-functional e-commerce system, meeting the requirements of comprehensive user, product, transaction and analytical reporting management. Through the implementation process, the system has demonstrated its ability to operate stably, flexibly and expand in practice. Specifically, the outstanding results include:

Build a detailed, secure user management and authorization system:

The system has completed core features such as account registration, login and logout with strict security mechanisms. User passwords are encrypted with a secure hash algorithm before being stored in the database, helping to protect personal data from the risk of attacks and unauthorized access.

User authorization is implemented precisely, clearly dividing three main groups: Administrators have the right to manage the entire system, Sellers have the right to manage their products and orders, and Customers have the right to place orders and manage personal accounts. This authorization mechanism both ensures safety and optimizes user experience.

The personal information management interface allows users to update data conveniently and securely, and provides a password change function to enhance account security.

Develop diverse and user-friendly product and category management modules:

The product management interface is designed to be intuitive and user-friendly, especially for sellers. Sellers can easily add, edit, delete or update product inventory.

The system supports product classification into many different categories, helping buyers easily search and filter suitable products according to personal needs or according to set criteria such as price, brand, or rating.

The database is designed to be standardized with tightly related tables, ensuring detailed, complete and synchronized information storage between entities such as products, categories and sellers. This contributes to increasing the efficiency of data retrieval and management throughout the system.

Manage and process sales transactions accurately and transparently:

The shopping cart system is built with high flexibility, allowing users to add products, change quantity or remove products from the cart before proceeding to checkout.

The payment function is fully integrated with steps to calculate costs incurred such as taxes, shipping fees and total amount payable, ensuring accuracy and transparency throughout the entire transaction process.

The system fully records and tracks order status from creation to completion, helping buyers and sellers easily manage, reconcile and promptly handle arising issues.

Detailed transaction history is stored, helping to support after-sales activities, complaint handling and accurate business statistics.

Deploy advanced, visual data reports and statistics:

The system uses modern charting libraries like JFreeChart to display sales reports, order counts, best-selling products, and seller performance as bar charts, pie charts, and line charts.

Users can customize reporting parameters such as time, product category or geographic area, thereby gaining a deeper and more comprehensive view of business operations.

The reporting feature is updated in real time, ensuring accurate and timely information, effectively supporting strategic decision making of the board of directors and related departments.

Extensive application of knowledge of Object Oriented Programming (OOP) and modern software technology:

The software architecture is designed according to a clear layered model, separating the interface layer, the business processing layer and the data access layer, helping to increase the ability to expand, maintain and upgrade in the future.

Using POJO classes to represent key entities such as User, Product, Order makes the data model intuitive and easy to manage.

The DAO (Data Access Object) layer is responsible for interacting directly with the database, hiding query details and optimizing performance.

The Service class handles complex business logic, adheres to the principles of encapsulation, inheritance and polymorphism in OOP, ensuring clean, maintainable and extensible source code.

The user interface is built with JavaFX combined with FXML, using SceneBuilder to create GUI in a structured, vivid and friendly way, supporting users to operate quickly and accurately.

Project management through Apache Maven ensures library management, project building and product distribution take place smoothly and professionally.

Effectively apply data structures in Java Collections Framework such as ArrayList, HashMap to store, process and manage temporary data in memory effectively.

Complete project structure and detailed technical documentation system:

The project directory structure is scientifically organized, clearly divided between source code, interface resources (FXML, CSS, images) and SQL scripts to create tables, insert sample data, making project development, management and maintenance convenient.

Detailed technical documentation including system design documents, installation instructions, configuration and software usage are carefully prepared, supporting the development team and end users to easily access and operate the system.

These documents also help ensure continuity during development, especially when multiple members are involved or when the project is transferred.

Proposed further development:

Expand integration of diverse payment methods: Implement support for online payment methods such as e-wallets, credit cards, and bank transfers to enhance user experience and increase convenience.

Mobile application development: Building applications on Android and iOS platforms helps users access the system anytime, anywhere, increasing flexibility and expanding customer base.

Apply artificial intelligence and advanced data analytics: Develop machine learning models to predict shopping trends, personalize product recommendations, optimize marketing campaigns, and improve business performance.

Integrated rating and feedback system: Improve product and seller rating functions, provide quick feedback channels to improve service quality and system reliability.

Enhance security and data protection: Apply multi-factor authentication (MFA) solutions, encrypt data in transit and at rest, regularly check and update security patches to protect the system against increasingly sophisticated attacks.

Improve user experience: Optimize the interface, integrate support chatbots, provide smart suggestions and upgrade search features to increase convenience and customer satisfaction.

**Summary:**

With the comprehensive completion of user management, product, transaction and analytical reporting functions, the Marketplace Transactional System project has created a solid foundation, meeting the needs of both sellers and buyers in the modern e-commerce environment. The system not only helps optimize transaction processes but also effectively supports business management and development in an effective, safe and transparent manner.

The project development process is also an opportunity to apply and improve professional knowledge in object-oriented programming, system design and software project management. The results achieved will be a solid foundation for further development of advanced features, expanding the scope of application in the future, contributing to promoting the development of the domestic and international e-commerce market.

**REFERENCES**

*[1]Oracle Corporation (2024), Java Platform, Standard Edition Documentation, accessible at: https://docs.oracle.com/en/java/javase/*

*[2]Apache Maven Project (2024), Introduction to the Apache Maven Build System, accessible at: https://maven.apache.org/guides/introduction/introduction-to-the-maven-build-system.html*

*[3]JFreeChart Developer Guide (2023), Creating Charts in Java, available at: https://www.jfree.org/jfreechart/*

*[4]MySQL Official Documentation (2024), MySQL 8.0 Reference Manual, accessible at: https://dev.mysql.com/doc/*

*[5]Bruce Eckel (2006), Thinking in Java, 4th Edition, Prentice Hall.*

*[6]Martin Fowler (2018), Patterns of Enterprise Application Architecture, Addison-Wesley Professional.*

*[7]Freeman, Eric & Robson, Elisabeth (2014), Head First Design Patterns, O'Reilly Media.*

*[8]Bui Minh Thanh (2023), Object-Oriented Programming with Java and JavaFX, National University Publishing House.*